Bachelorarbeit

Titel:

Analyse, Design und Umsetzung eines RAD-Framework für mobile Geräte

Forschungsfrage:

Können mit dem bestehenden Path-Framework hybride Applikationen entwickelt werden, die sowohl auf dem Desktop und dem Smartphone funktionieren und zudem benutzerfreundlich und verständlich für den Endnutzer sind?

Institution:	ZHAW School of Management and Law
Modul:	Bachelor Thesis
Betreuung:	Adrian Moser
Datum der Abgabe:	24. Mai 2018

Autor:	Manuel Weber (15-540-057)
	Untere Aegerten 9, 8143 Stallikon
	weberm16@students.zhaw.ch
	manuel_weber@bluewin.ch

Management Summary

Unternehmen im digitalen Zeitalter benötigen verschiedene Applikationen, welche die Abwicklung von Geschäftsprozessen mit internen und externen Partnern unterstützen. Um an solche Anwendungen zu gelangen, haben Unternehmen drei Möglichkeiten: Standardlösungen, Entwickeln durch externe Anbieter und Eigenentwicklung. Eine Eigenentwicklung wird von den meisten Unternehmen bevorzugt, jedoch ist eine solche für viele Firmen schwierig umzusetzen. Gründe dafür sind das fehlende Wissen in Unternehmen und der grosse Aufwand der Entwicklung. Rapid Application Development (RAD) beschäftigt sich mit dem möglichst schnellen und einfachen Entwickeln von Applikationen. Um dieses Ziel zu erreichen, gibt es verschiedene RAD-Frameworks, die einen Teil, oder sogar den ganzen Prozess, einer Applikationsentwicklung vereinfachen und beschleunigen. In einem Forschungsprojekt der ZHAW ist das Path-Framework entstanden. Dieses Framework unterstützt das Entwickeln des Front-End durch das Erstellen eines technologieunabhängigen GUI (Graphical User Interface) für Desktopcomputer. Durch das massive Zunehmen von mobilen Endgeräten müssen Applikationen heutzutage nicht mehr nur für den Desktop, sondern auch für Smartphones und Tablets entwickelt werden. Aus diesem Grund wird das Path-Framework in dieser Arbeit weiterentwickelt, sodass damit ein generisches GUI definiert werden kann, das auf verschiedenen Endgeräten optimiert angezeigt wird. Fragen, die in diesem Zusammenhang beantwortet werden müssen, sind, ob es überhaupt möglich ist, mit dem Path-Framework sogenannte hybride Applikationen zu entwickeln, und ob solche Applikationen benutzerfreundlich und verständlich für den Endnutzer sind.

Um diese Fragen zu beantworten, wurde in einem ersten Schritt ein theoretischer Hintergrund zum Thema RAD durch Literatur-Review erarbeitet. Danach wurde das Path-Framework weiterentwickelt und damit eine Beispielapplikation erstellt. Mit dieser Beispielapplikation wurde evaluiert, ob mit dem Path-Framework hybride Anwendungen erstellt werden können. Weiter wurden Usability Tests mit verschiedenen

Testpersonen durchgeführt, um die Benutzerfreundlichkeit und die Verständlichkeit einer Path-Applikation abzuklären. Die Beispielapplikation, die entwickelt wurde, hat gezeigt, dass es möglich ist, mit dem Path-Framework hybride Applikationen zu erstellen. Die App wurde als Desktopversion über den Browser und als native App über den Google Play Store zur Verfügung gestellt. Das Ergebnis der Usability Tests bestätigte zudem, dass die Applikation von Endnutzern als benutzerfreundlich wahrgenommen wird und einfach zu bedienen ist.

Das Fazit dieser Arbeit ist, dass es möglich ist, mit RAD-Tools die Applikationsentwicklung bedeutend zu vereinfachen und zu beschleunigen. Es ist möglich hybride Applikationen, die einmal programmiert werden müssen und dann auf verschiedenen Geräten optimiert angezeigt werden, mit RAD-Frameworks zu erstellen. Aus diesem Grund ist zu empfehlen, das Path-Framework weiterzuentwickeln. In welche Richtung und in welchem Ausmass es weiterentwickelt werden soll, muss nun entschieden werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung und Problemstellung

Heutzutage gibt es in jedem Unternehmen eine Vielzahl von IT-Applikationen, die für die Abwicklung von Geschäftsprozessen mit internen und externen Partnern genutzt werden. Keine Firma kann mittlerweile ohne solche Software überleben. Um an Geschäftssoftware zu gelangen, haben Unternehmen drei Möglichkeiten: Standardlösungen, Entwicklung durch externe Anbieter und Eigenentwicklung. Es gibt zahlreiche Standardlösungen auf dem Markt. Diese sind meistens preiswert und decken einen Grossteil der Prozesse ab, die in den meisten Unternehmen vorkommen. Häufig muss die Standardsoftware aber mit Eigenentwicklungen ergänzt werden, da das Standardpaket nicht ausreicht oder zu grosse und aufwändige Anpassungen vorgenommen werden müssten. Das Entwickeln von unternehmenseigener Software durch einen Drittanbieter ist kostenintensiv. Ein weiteres Problem dabei ist, dass man sich für längere Zeit an einen Anbieter binden muss. Denn die Geschäftsbeziehung ist mit dem Entwicklungsprozess nicht abgeschlossen. Im Normalfall übernimmt dieser Anbieter auch die Wartung der Software. Eine Eigenentwicklung ist für die meisten Unternehmen schwierig umzusetzen, da der Entwicklungsprozess mit sehr hohem Aufwand verbunden ist. Zudem müssen Wartung und Changemanagement meistens trotzdem einem externen Anbieter überlassen werden, da dies schnell komplex und aufwändig wird. Das führt wiederum zu einem Kontrollverlust und Abhängigkeiten. Eine weitere Herausforderung für die Eigenentwicklung stellt die enorme Zunahme an mobilen Endgeräten in den letzten zehn Jahren dar. Folglich müssen heute Geschäftsapplikationen nicht nur für den Desktop, sondern auch für das Smartphone und das Tablet zur Verfügung stehen.

Ein Hilfsmittel für die Eigenentwicklung sind Rapid Application Development Frameworks. Dies sind IT-Werkzeuge, die das Entwickeln von Applikationen vereinfachen und zudem die Time-to-Market von Applikationen verkürzen. Ein solches Framework ist in einem Forschungsprojekt der ZHAW entstanden. Es trägt den Namen 'Path-Framework' und stellt ein technologieunabhängiges GUI (Graphical User Interface) zur Verfügung, das von einer Engine als Webapplikation dargestellt wird. Das heisst, dass das Path-Framework das Programmieren des Front-End übernimmt

und somit den Entwicklungsaufwand reduziert. Das Front-End einer Path-Applikation wird in einem JSON-Dokument definiert. In diesem Dokument kann mithilfe von vordefinierten Elementen die Benutzeroberfläche für die Applikation erstellt werden. Abbildung 1 illustriert die Funktionsweise vom Path-Framework. Das GUI-Modell, welches als JSON definiert ist, wird von einer Engine gerendert und dann als webbasierte Applikation mit HTML5 und JavaScript zur Verfügung gestellt. Das Path-Front-End ist technologieunabhängig und kann somit mit einem beliebigen Back-End ergänzt werden.

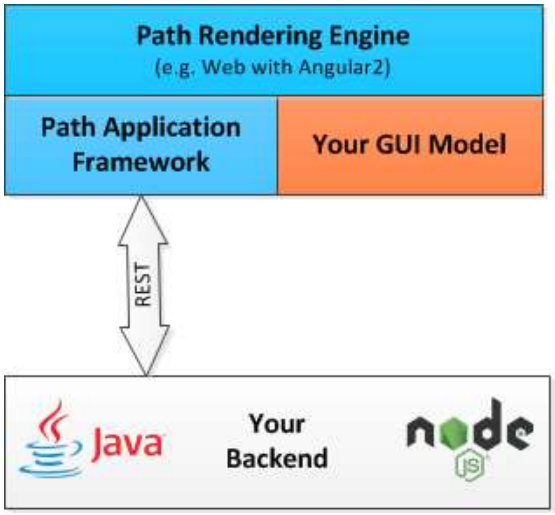


Abbildung 1: Aufbau Path-Framework (Quelle: https://github.com/innovad/path)

Das Path-Framework erzeugt ein GUI für Desktop Computer. In dieser Arbeit wird das Path-Framework weiterentwickelt, sodass damit hybride Applikationen erstellt werden können. Das heisst, dass das GUI, das mit dem Path-Framework erzeugt wird, je nach Bildschirmgrösse des Endgerätes optimiert angezeigt wird. Die Forschungsfrage für diese Arbeit lautet:

> Können mit dem bestehenden Path-Framework hybride Applikationen gebaut werden, die sowohl auf dem Desktop und dem Smartphone funktionieren und zudem benutzerfreundlich und verständlich für den Endnutzer sind?

Nach dem Weiterentwickeln des Path-Framework wird eine Beispielapplikation damit erstellt, die zeigen soll, dass es möglich ist, mit dem Path-Framework hybride Applikationen zu entwickeln. Diese Beispielapplikation wird als Webapplikation und als native App für Android dargestellt. Anhand von der Beispielapplikation werden Usability Tests durchgeführt, mit denen validiert wird, ob eine Applikation mit dem Path-Framework von Endnutzern als benutzerfreundlich und verständlich wahrgenommen wird.

2 Aufbau und Schwerpunkte der Arbeit

In diesem Teil werden der Aufbau und die Schwerpunkte der Arbeit dargelegt. Die Arbeit ist so aufgebaut, dass zuerst ein theoretischer Hintergrund für Rapid Application Development dargelegt wird. Dann werden verschiedene bestehende Rapid Application Development (RAD) Frameworks aufgezeigt und mit dem Path-Framework verglichen. Damit wird gezeigt, wie das Path-Framework im Vergleich zu anderen Lösungen einzuordnen ist. Zudem werden aus der Analyse von bestehenden RAD Frameworks Erweiterungsvorschläge für das Path-Framework herausgearbeitet. Im darauffolgenden Teil wird die Weiterentwicklung des Path-Framework, das Erstellen der Beispielapplikation und das Veröffentlichen dieser Applikation dokumentiert. Danach folgt eine Anleitung, wie aus einer Path Webapplikation eine hybride App erstellt werden kann. Abschliessend wird das Ergebnis aus den Usability Tests und somit die Validierung der Forschungsfrage präsentiert. Die Schwerpunkte der Arbeit liegen auf dem Weiterentwickeln und dem Validieren des Path-Framework. Ziel ist es, das Path-Framework soweit weiterzuentwickeln, dass damit hybride Applikationen erstellt werden können und damit eine Beispielapplikation zu entwickeln, mit deren Hilfe validiert werden kann, ob es möglich ist, mit diesem Framework hybride Applikationen zu erstellen, die benutzerfreundlich und verständlich sind.

3 Methodik

In diesem Kapitel werden das Vorgehen und die verwendeten Methoden dieser Bachelorarbeit aufgezeigt. Die Ausgangslage ist das bestehende Path-Framework der ZHAW. Ausgehend von diesem Tool wird vertieft auf das Thema Rapid Application Development eingegangen. Es wird ein theoretischer Hintergrund über das Thema gegeben und verschiedene, bestehende Lösungen vorgestellt. Das Path-Framework wird mit den bestehenden Lösungen verglichen. Danach wird eine Beispielapplikation erstellt, mit welcher schliesslich die Forschungsfrage validiert wird. Für das Erarbeiten dieser Bachelorarbeit werden verschiedene wissenschaftliche Methoden angewendet. Für den Theorieteil wird primär die Methode Literatur-Review angewandt. Das Erstellen der Beispielapplikation kann als Experiment betrachtet werden und für die Usability Tests werden die Methoden Umfrage und Beobachtungen verwendet. Die Arbeit wird in drei verschiedene Phasen aufgeteilt: Vorbereitung, Path-Erweiterung und Abschluss. Abbildung 2 zeigt in welchem Zeitraum die verschiedenen Arbeitspakete bearbeitet werden.

	Zeitrahmen in Wochen	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Phase Vorbereitung															
Literaturrecherche und State of Art	13														
Mock-Ups	3														
div. Vorbereitungsarbeiten	3														
Disposition	3														
Phase Path-Anpassung															
Bootstrap	7														
Path-Optimierung für mobile Geräte	7														
Anleitung hybride Apps	5														
Beispielapplikation	5														
Phase Abschluss															
Usability-Tests	2														
Management Summary	1														

Abbildung 2: Gantt-Diagramm Ablauf Bachelorarbeit

Da die Arbeit drei verschiedenartige Teile (Theorieteil, Entwicklung und Usability Tests) enthält, wird das Kapitel Methodik aufgeteilt. Somit ist in den Kapiteln Dokumentation und Usability Tests je ein Unterkapitel Methodik angesiedelt.

4 Theoretischer Hintergrund

Im Kapitel 'Theoretischer Hintergrund' wird der Begriff Rapid Application Development erläutert. Es wird aufgezeigt von wo der Begriff kommt und inwiefern er sich von klassischen Entwicklungsmodellen unterscheidet.

Rapid Application Development (RAD) ist eine Entwicklungsmethode für Informationssysteme. Diese Entwicklungsmethode dient dazu Applikationen möglichst schnell und kostengünstig zu entwickeln, ohne dabei an Qualität einzubüssen (Hirschberg, 1998, S. 1). Beynon-Davies, Mackay & Tudhope (2000, S. 195) zeigen auf, dass Rapid Application Development sich durch ein iteratives Prototyping und ein hohes Level an Einbezug von Endnutzern, sowie anderen Stakeholdern, auszeichnet. RAD verwendet Techniken wie 'Mixed Development Teams' (auch Joined Development Teams), produktorientiertes Projektmanagement und inkrementelles und iteratives Prototyping (Beynon-Davies et al., 2000, S. 195). Mixed Development Teams bezeichnet hierbei ein Zusammenarbeiten von Entwicklern, Endnutzern und Mitarbeitern aus dem operativen Geschäft eines Unternehmens (Beynon-Davies et al., 2000, S. 195).

Berger, Beynon-Davies & Cleary (2004, S. 3) sehen die Anfänge von Rapid Application Development im Jahr 1991. In diesem Jahr wurde die Entwicklungsmethode erstmals von James Martin formalisiert (Martin, 1991). Er verstand darunter einen Entwicklungslebenszyklus, der schneller und günstiger als traditionelle Entwicklungslebenszyklen war (Berger et al., 2004 S. 3). Mitte der Neunzigerjahre wurde der Begriff Rapid Application Development jedoch bereits zu einem Sammelbegriff, der verschiedene Methoden, Techniken und Tools von verschiedensten Anbietern und Forschern zusammenfasst (Berger et al., S.3). Bis heute ist Rapid Application Development ein breit gefasster Begriff, der keine präzise Definition kennt.

Hirschberg (1998, S. 1) nennt als Grundidee von RAD die Softwareentwicklung im Vergleich zu klassischen Vorgehensmodellen, wie zum Beispiel dem Wasserfallmodell, flexibler, schneller und günstiger zu gestalten. Dies wird durch eine iterative Vorgehensweise erreicht. Somit werden die Phasen der Analyse, des Designs, des

Programmierens und des Testens im Entwicklungszyklus in kurze iterative Entwicklungszyklen aufgeteilt (Abbildung 3). Während beim Wasserfallmodell jeder Task erst nach dem Abschliessen des Vorherigen begonnen wird, findet dies beim RAD iterativ statt. Das heisst, dass das Entwickeln in vielen kurzen Entwicklungszyklen stattfindet, in denen jeweils ein kleines Stück Software gebaut wird. Dieses Stück Software wird dann getestet und sogleich dem Endnutzer vorgelegt. Von diesem wird direkt Feedback eingeholt und anhand von diesem Feedback wird das gebaute Stück Software optimiert und auf die Wünsche und Vorstellungen des Endnutzers angepasst. Diese Entwicklungszyklen werden auch 'Build-Execute-Modify' Loops genannt (Hirschberg, 1998, S. 4). Diese Loops finden während der gesamten Softwareentwicklung statt, so dass immer wieder ein kleiner Teil mehr dazukommt, der sogleich technisch und vom End User getestet wird (Hirschberg, 1998, S. 4).

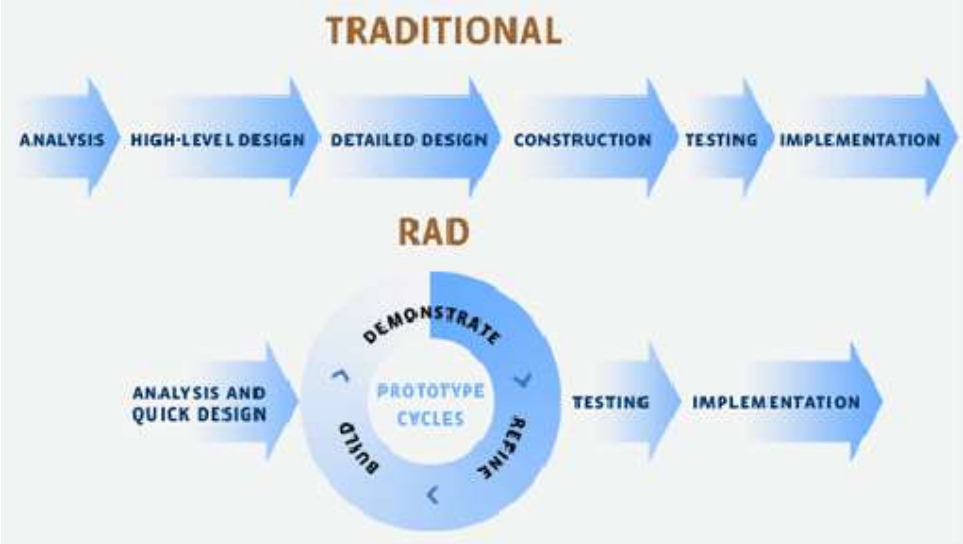


Abbildung 3: Wasserfallmodell vs. RAD (Biesemans, Horsten & Deroose, 2010)

Grundlage des Rapid Application Development ist das Spiralmodell, das 1988 von Barry W. Boehm entwickelt wurde (Abbildung 4). Das Spiralmodell veranschaulicht einen agilen und iterativen Softwareentwicklungsprozess (Ruparelia, 2010, S. 10). Boehm (1988) modifizierte das Wasserfallmodell, indem er Zyklen darin einbaute. Er versuchte damit zwei Hauptschwierigkeiten des Wasserfallmodells zu lösen. Erstens kannte das klassische Wasserfallmodell zwei Designphasen. Dies war in gewissen Situationen aber nicht notwendig und teilweise sogar überflüssig, wie Ruparelia (2010, S. 10) bemerkt. Zweitens war der Top-Down Ansatz für das Entwickeln von Software

aufgrund der sich immer schneller verändernden Unternehmensumwelt nicht mehr zeitgemäss (Ruparelia, 2010, S. 10). Ruparelia (2010, S. 10 & 11) sieht im Spiralmodell einen Paradigmenwechsel von einem spezifikationsgetriebenen zu einem risikogetriebenen Vorgehensmodell. Das Spiralmodell besteht aus vier Quadranten. Diese vier Quadranten sind: Festlegen der Ziele, Beurteilen von Alternativen/Risikoanalyse, Entwicklung und Test, sowie Planung des nächsten Zyklus'. Jeder dieser vier Quadranten wird bei jeder Iteration durchlaufen.

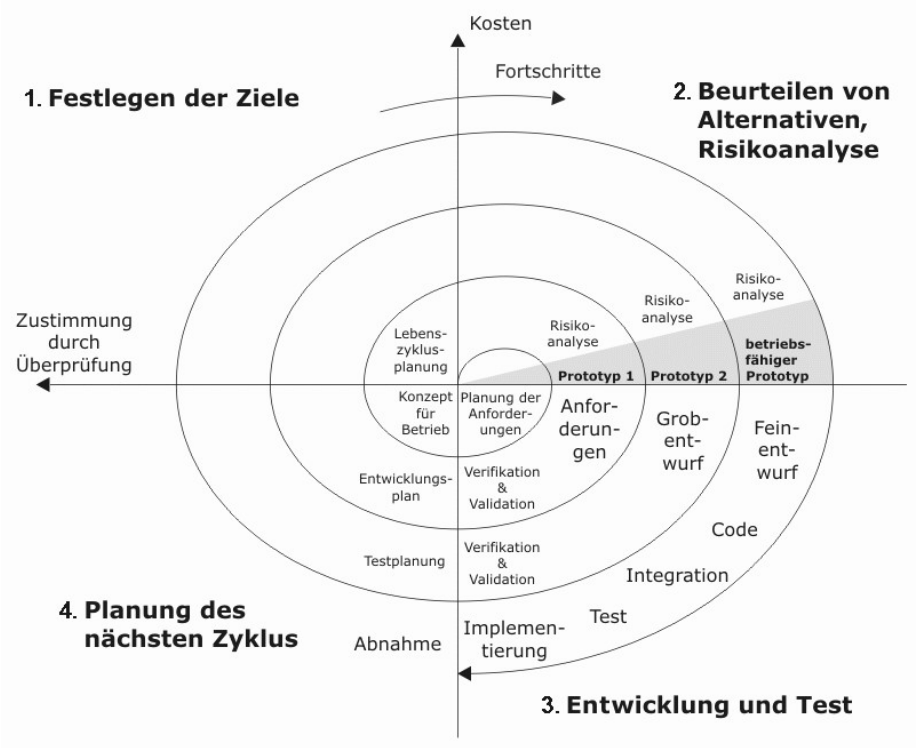


Abbildung 4: Spiralmodell (Boehm, 1988)

Gemäss Beynon-Davies et al. (2000, S. 203) gab es zwei Hauptgründe für die Notwendigkeit von Rapid Application Development: die Betriebsunsicherheit (Business uncertainty) und die Entwicklungsunsicherheit (Development uncertainty). Unter der Betriebsunsicherheit wird die schnelllebige Unternehmensumwelt verstanden (Beynon-Davies et al., 2000, S. 203). Dadurch müssen Unternehmen schnell auf Veränderungen in ihrem Umfeld reagieren können um zu überleben (Beynon-Davies et al., 2000, S. 203). Als Beispiele für schnelle Veränderungen im Unternehmensumfeld um die Jahrtausendwende nennen Beynon-Davies et al. (2000, S. 203) die Globalisierung der Märkte und der Produktionsstätten, virtuelle Organisationen und auf

Kundenwünsche anpassbare Produkte. Die Entwicklungsunsicherheit beschreibt, dass immer mehr Technologien und Tools zum Entwickeln von Applikation zur Verfügung stehen (Beynon-Davies et al., 2000, S. 203). Auch heutzutage ist die Betriebsunsicherheit auf Grund der Digitalisierung weiterhin ein wichtiger Punkt. Und auch die Entwicklungsunsicherheit, in Bezug auf die zur Verfügung stehenden Technologien und Tools, wird immer grösser. Aus diesem Grund ist der Softwareentwicklungstrend Rapid Application Development auch in den letzten Jahren wieder vermehrt thematisiert worden (Franklin, 2016).

Während früher beim Thema Rapid Application Development in erster Linie vom Entwickeln von Informationssystemen und heute vom Entwickeln von Apps gesprochen wird, meint beides jedoch im Grunde das Gleiche. Es geht um das Entwickeln von Anwendungen, die dem Endnutzer Daten zur Verfügung stellen und so einen Mehrwert generieren. Zudem geht es darum diese Anwendungen so schnell, so gut und so günstig wie möglich zu entwickeln. Heutzutage gibt es viele sogenannte RAD-Tools oder RAD-Frameworks. Darunter sind Bausätze für Applikationen zu verstehen, die es einem ermöglichen, möglichst einfach Applikationen zu erstellen. Da es aber sehr viele verschiedene Arten von Applikationen gibt, gibt es auch zahlreiche RAD-Tools. Die meisten RAD-Tools verwenden visuelle Interfaces, die durch Drag-and-Drop bedient werden können (Franklin, 2016). Somit können Applikationen erstellt werden, für die nur wenig, oder sogar gar kein, Code geschrieben werden muss (Franklin, 2016). Aus diesem Grund spricht man heute in Bezug auf Rapid Application Development auch von no-code respektive low-code Softwareentwicklung (Franklin, 2016). Das funktioniert, weil der Code im Hintergrund automatisch erzeugt wird (Franklin, 2016). Somit werden zusätzlich zur Zeit- und Kosteneinsparung typographische Fehler minimiert und die Applikationsentwicklung wird auch Usern mit wenig oder gar keinen Programmierkenntnissen ermöglicht (Franklin, 2016).

Im nächsten Kapitel werden verschiedene RAD-Tools vorgestellt und miteinander verglichen. Zudem werden die fünf Tools evaluiert, die dem Path-Framework vom Aufbau her am ähnlichsten sind. Auf diese wird im Anschluss noch detaillierter eingegangen.

5 Vergleich Rapid Application Development Tools

Applikationen bestehen grundsätzlich aus mindestens drei Komponenten. Namentlich sind das ein User Interface, welches die Bedienungsoberfläche für Benutzer darstellt, eine Datenbank, die Informationen enthält und zur Verfügung stellt, und Geschäftsregeln (Business Rules / Workflow), welche die ersten zwei Komponenten verbinden (Franklin, 2016). Rapid Application Development Tools sind so aufgebaut, dass sie mit einer dieser drei Komponenten beginnen und dann die Features und Funktionen der anderen Komponenten hinzufügen. Somit ergeben sich drei verschiedene Ansätze von RAD-Tools: Interface-first, Database-first und Workflow-first.

Interface-first Development Tools werden gemäss Franklin (2016) häufig, aber nicht ausschliesslich, für die Entwicklung von Applikationen für mobile Endgeräte verwendet. Hier können User zuerst ihr Interface erstellen, dieses testen und dann Geschäftsregeln und eine Datenbank an das Userinterface anhängen. Das Designen und Programmieren von Datenbanken kann sehr komplex sein. Aus diesem Grund beginnen Database-first Tools mit dem Erstellen einer graphischen Repräsentation von den Datenstrukturen der Applikation. Der Workflow (Geschäftsregeln) und das Userinterface werden dann um die Datenbank herum gebaut. Workflow-first Frameworks beginnen mit dem Modellieren von (Geschäfts-) Prozessen. Das heisst, dass zuerst die Entscheidungspunkte, die Verarbeitungsregeln und –funktionen, sowie die Outcomes modelliert werden. Und erst danach werden das Interface und die Datenbank entwickelt.

Eine weitere Unterscheidung von RAD-Tools ist, ob die entwickelte Applikation eine native App auf einem gewählten Betriebssystem wird oder eine JavaScript und HTML5 Applikation, die in einem Browser oder in einem Wrapper auf den jeweiligen Betriebssystemen läuft (Franklin, 2016). Aus einer funktionalen Sichtweise, macht diese Unterscheidung keinen grossen Unterschied. Die User Experience kann jedoch stark variieren und die Performanz von nativen Applikationen ist besser.

Franklin (2016) zeigt auf, dass eine letzte Unterscheidung bei der Menge an Code, die geschrieben werden muss, gemacht wird. Es gibt Frameworks, bei welchen so gut wie kein Programmcode geschrieben werden muss, um eine funktionstüchtige Applikation zu bauen, während bei anderen immer noch eine ziemliche Menge Code benötigt wird.

5.1. Übersicht von RAD-Tools

Da es so viele verschiedene Arten von RAD-Tools gibt, existieren auch unzählige RAD-Frameworks. Eine nicht abschliessende Liste dazu ist die folgende:

- 4th Dimension
- ActiveTCL
- Alpha Software
- Apache ISIS
- Caspio
- Django
- Filemaker, Inc.
- KiSSFLOW – BPM & Workflow Software
- Lotus Domino Designer Client (IBM Notes)
- Mendix
- Ninox
- Nintex Workflow Platform
- Omnis
- Oracle Application Express
- OutSystems
- Pegasystems
- Quick Base
- Salesforce Lightning
- ServiceNow
- Visual Lansa
- VINYL

- XDEV
- Zoho Creator

Die bestehenden RAD-Tools werden in der folgenden Tabelle miteinander verglichen. Danach werden die fünf Tools, die dem Path-Framework am ähnlichsten sind, in einem weiteren Abschnitt detaillierter beschrieben und mit dem Path-Framework verglichen. Das Path Framework funktioniert nach dem Ansatz Interface-first, es erstellt eine auf JavaScript und HTML5 basierte Applikation und es muss Code geschrieben werden, jedoch nicht allzu viel (low-Code). Basierend auf diesen drei Punkten, wurden die RAD-Frameworks miteinander verglichen:

RAD-Tool	Ansatz	Nativ/Web	Code
4th Dimension	Database-first	Nativ	Code
ActiveTCL	Interface-first	Nativ	Code
Alpha Software	Interface-first	Webbasiert, nativ-like UI	Low-Code
Apache ISIS	Workflow-first	Webbasiert	Code
Caspio	Database-first	Webbasiert	Low-Code
Django	Interface-first	Webbasiert	Code
Filemaker, Inc.	Database-first	Nativ	Low-Code
KiSSFLOW – BPM & Workflow Software	Workflow-first	Nativ	No-Code

Tabelle 1 (Teil 1): Vergleich RAD-Tools

Lotus Domino Designer Client (IBM Notes)	Interface-first / Database-first	Webbasiert	Low-Code
Mendix	Interface-first	Webbasiert	Low-Code/No-Code
Ninox	Database-first	Nativ	No-Code
Nintex Workflow Platform	Workflow-first	Nativ	No-Code
Omnis	Interface-first	Webbasiert	Low-Code
Oracle Application Express	Database-first	Webbasiert	Low-Code
OutSystems	Interface-first	Webbasiert/Nativ	Low-Code
Pegasystems	Interface-first	Nativ	Low-Code
Quick Base	Interface-first	Nativ	Low-Code / No-Code
Salesforce Lightning	Interface-first	Nativ	Low-Code
ServiceNow	Database-first	Nativ	Code
Visual Lansa	-	Web/Nativ	Low-Code
VINYL	Database-first	Webbasiert	No-Code
XDEV	Interface-first	Nativ	Code
Zoho Creator	Interface-first	Webbasiert	Low-Code

Tabelle 1 (Teil 2): Vergleich RAD-Tools

5.2 Beschrieb der selektierten RAD Tools

Die fünf RAD-Tools, die dem Path-Framework am ähnlichsten sind, wurden in der Vergleichstabelle (siehe Tabelle 1) grün hinterlegt. Diese werden in diesem Abschnitt genauer beschrieben und einzeln mit dem Path-Framework verglichen. Dann werden Vor- und Nachteile des Path-Framework gegenüber diesen fünf RAD-Tools aufgezeigt. Zudem werden mögliche Erweiterungen für Path aus den analysierten RAD-Tools genannt.

5.2.1 Alpha Software

Alpha Software ist eine low-code Plattform für Rapid Development, Distribution und Deployment von Applikationen. Es ist auf das Entwickeln von mobilen, webbasierten Geschäftsapplikationen ausgelegt und erstellt hybride Applikationen, die einmal geschrieben werden müssen und dann auf allen Endgeräten zur Verfügung gestellt werden können. Die hybriden Apps verfügen über ein 'nativ-like' Userinterface und haben Zugriff auf die gerätespezifischen Funktionen, wie zum Beispiel der Kamera oder dem GPS. Zudem wird durch das Einsetzen von verschiedenen Wizards und anderen Tools auch Mitarbeitern mit beschränkten Programmierkenntnissen das Entwickeln von Applikationen ermöglicht (Alpha Software, 2018).

Im Vergleich zum Path-Framework, das in einem Forschungsprojekt an der ZHAW entstanden ist, ist Alpha Software ausgereifter und machtvoller. Zudem unterstützt es auch die Entwicklung von einem Back-End. Ein Vorteil, der das Path-Framework gegenüber Alpha Software aufweist, ist das technologieunabhängige GUI. Das heisst, dass das Back-End unabhängig vom Path-Front-End entwickelt werden kann. Ein weiterer Vorteil vom Path-Framework ist, dass es gratis ist und der Code jedem über GitHub zur Verfügung steht.

5.2.2 Lotus Domino Designer Client (IBM Notes)

Der Lotus Domino Designer unterstützt die Entwicklung von Applikationen für die Domino-Plattform mithilfe von XPages, Formularen, Ansichten und weiteren Elementen. XPages ist eine zeiteffiziente Anwendungsentwicklungstechnologie für die Erstellung bereichsübergreifender, webbasierter Anwendungen. Es baut auf der Technologie von Java Server Faces auf und ermöglicht Daten von IBM Notes und relationalen Datenbanken über Browserclients auf allen Betriebssystemen anzuzeigen. XPages basiert auf Webentwicklungs-Programmiersprachen. Die Applikationen können als dynamische Web 2.0-Seiten sowohl in einem Browser als auch im Notes-Client bereitgestellt werden. Die Entwicklungsumgebung stellt eine Drag-and-drop-Entwurfsumgebung sowie eine Umgebung für XML-Quellcode zur Verfügung. Das Layout wird über CSS und die Geschäftslogik über JavaScript gesteuert (IBM, 2018).

Auch hier muss gesagt werden, dass es sich um ein professionell entwickeltes Framework von IBM handelt und es aus diesem Grund ein mächtigeres Instrument darstellt als das Path-Framework. Aber auch hier ist die Unabhängigkeit von einem grossen Anbieter und die Technologieunabhängigkeit betreffend Back-End ein Vorteil von Path.

5.2.3 Mendix

Mendix wurde im Jahr 2005 gegründet. Es ist eine low-code Software-Plattform, die Tools zum Entwickeln, Testen, und Warten von Applikationen zur Verfügung stellt. Mendix Apps können mithilfe von einem visuellen Editor erstellt werden. Das heisst die Apps können mit Drag-and-Drop erstellt werden. Dies ermöglicht es sogar Mitarbeitern aus operativen Abteilungen bei der Entwicklung von Apps mitzuhelfen, da dazu kein Code geschrieben werden muss. Der visuelle Editor fungiert zusätzlich als Übersetzungssprache zwischen den Mitarbeitern aus der IT und denjenigen aus den operativen Bereichen. Was Mendix zusätzlich auszeichnet ist, dass nicht nur das Entwickeln von Apps angeboten wird, sondern ein ganzes Application Lifecycle Management System, das auch die Wartung und Weiterentwicklung von bestehenden

Applikationen ermöglicht. Auch mit Mendix können hybride, responsive Apps erstellt werden, die sich auf jedem Gerät so verhalten, als wären sie direkt für das jeweilige Betriebssystem geschrieben worden (Mendix, 2018).

Mendix ist eine gesamte Entwicklungsplattform und deckt einen viel breiteren Bereich ab als das Path-Framework, das nur auf das Entwickeln vom Front-End ausgelegt ist. Aber auch hier ist die Technologieunabhängigkeit für das Back-End ein wichtiger Punkt. Zudem ist man mit dem Path-Framework an keine Plattform und somit an kein anderes Unternehmen gebunden. Eine Erweiterungsidee für das Path-Framework wäre hier, ebenfalls einen visuellen Editor für die Front-End-Programmierung zur Verfügung zu stellen. Somit könnte die JSON-Datei, die im Path-Framework für das Erzeugen des Front-End zuständig ist, im Hintergrund aus dem visuellen Editor erzeugt werden.

5.2.4 Omnis Studio

Omnis Studio ist das Framework, das Path am ähnlichsten ist. Es ist auch für das Erstellen des GUI, also für das Erstellen der Benutzeroberfläche, zuständig. Das Erstellen des GUI erfolgt visuell per Drag-and-Drop. Omnis Studio enthält eine gesamte IDE (integrierte Entwicklungsumgebung) mit einer integrierten Versionsverwaltung. Das GUI, das mit Omnis Studio erstellt wird, ist ebenfalls technologie- und plattformunabhängig und die erstellten Applikationen können mit Hilfe eines mitgelieferten Wrappers als iOS und/oder Android Applikationen kompiliert werden (Omnis, 2018).

Omnis Studio verfolgt den gleichen Ansatz wie das Path-Framework: Das Entwickeln eines technologieunabhängigen GUI, damit das Front-End einer Applikation mit weniger Aufwand erstellt werden kann. Omnis bietet out-of-the-box mehr an als das Path-Framework. Jedoch kann das Path-Framework mit anderen Tools und Frameworks kombiniert werden um eine ähnliche Leistungspalette abzudecken. Ein grosser Vorteil von Omnis Studio ist die visuelle Erstellung des Front-End. Diese Funktion kann als Erweiterungspotential für das Path-Framework betrachtet werden. Ein

Nachteil von Omnis Studio ist, dass man auf die mitgelieferte IDE zurückgreifen muss und aus diesem Grund eingeschränkter ist als mit dem Path-Framework.

5.2.5 OutSystems

OutSystems ist ebenfalls eine low-code Plattform. Mit OutSystems kann die gesamte App visuell entwickelt und existierende Systeme können integriert werden. Des Weiteren besteht die Möglichkeit die Applikation mit Code zu ergänzen, falls dies gewünscht ist. Auch mit OutSystems können hybride Apps entwickelt werden, die sich verhalten, als wären sie für das jeweilige Betriebssystem geschrieben worden. Zudem ist es möglich das Back-End und das Front-End unabhängig voneinander zu entwickeln. Das heisst, dass nicht beide Komponenten mit OutSystems programmiert werden müssen. Der Fokus von OutSystems liegt vor allem auf der Geschwindigkeit. Gemäss eigenen Angaben des Unternehmens ist es mit OutSystems möglich auch komplexe Applikationen in einem Zeitraum von 3 bis 4 Monaten zu entwickeln. OutSystems bezeichnet sich selber als Marktführer im Bereich Rapid Application Development (OutSystems, 2018).

OutSystems deckt den ganzen Entwicklungsprozess von Applikationen ab und ist deshalb dem Path-Framework in allen Hinsichten überlegen. Der einzige Vorteil, den das Path-Framework bietet ist, dass es gratis erhältlich und somit auch für Startups oder Privatpersonen verfügbar ist. Erweiterungsmöglichkeiten können von OutSystems verschiedene abgeleitet werden. Beispiele dafür sind das visuelle Anlegen von GUI's und das visuelle Abbilden der Geschäftslogik.

6 Dokumentation

Der Grossteil dieser Bachelorarbeit war das Schreiben von Programmiercode. Im Verlauf der Arbeit wurde das Path-Framework weiterentwickelt, eine Beispielapplikation inklusive kleinem Back-End erstellt, diese Beispielapplikation wurde als mobile, native Applikation zur Verfügung gestellt und schlussendlich über den Google Play Store veröffentlich. In diesem Kapitel wird zuerst die Methodik für den Entwicklungsteil erläutert und dann werden das Vorgehen, die Meilensteine, die Problempunkte und die Ergebnisse der Entwicklung aufgezeigt.

6.1 Methodik

In diesem Abschnitt wird das Vorgehen und die Methode für die Weiterentwicklung des Path-Framework und der Entwicklung der Beispielapplikation dargelegt. Entwickelt wird in kurzen, iterativen Zyklen. Das heisst, dass kleine Änderungen am Code vorgenommen werden, die dann sofort getestet werden. Getestet wird über das Anzeigen der Änderungen in einem Browserfenster. Die wissenschaftliche Methode für die Entwicklung ist 'Experiment', da damit getestet wird, ob mit dem Path-Framework hybride Applikationen erstellt werden können. Die Dokumentation während der Entwicklung wird über GitHub gepflegt.

6.2 Weiterentwicklung Path-Framework für mobile Geräte

6.2.1 Allgemeine Anpassungen

Das Path-Framework wurde and der ZHAW School of Management and Law entwickelt. Nun sollte es soweit weiterentwickelt werden, dass damit hybride Applikationen erstellt werden können. Hybride Applikationen sind Anwendungen, die einmal programmiert werden müssen und dann auf verschiedenen Geräten und verschiedenen Betriebssystemen funktionieren und sich verhalten, als wären sie für die jeweilige Plattform programmiert worden. Die Ausgangslage war, dass das Path-Framework anhand eines JSON-Dokuments ein GUI (Graphical User Interface) erstellen konnte,

jedoch eines, das nur für Desktop-Computer ausgelegt war. Das heisst, dass das Path-Framework soweit weiterentwickelt werden musste, dass es auch ein GUI für Smartphones erzeugen kann.

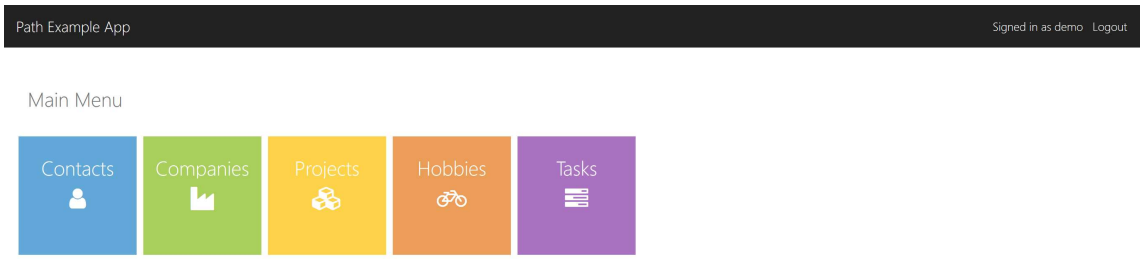


Abbildung 5: Ausgangslage Entwicklung Path-Framework

Der erste Schritt im Entwicklungsprozess war der Umstieg vom CSS-Framework 'metro-bootstrap' auf das populäre Framework 'Bootstrap'. Der Grund für diesen Umstieg war, dass 'metro-bootstrap' nicht mehr weiterentwickelt wird. Ziel war, dass das Path-Framework nach der Umstellung des CSS-Framework möglichst ähnlich aussah wie zuvor. Nach dem 'Umhängen' auf Bootstrap gab es bereits ein erstes Problem: Die sogenannten Tiles, die gut die Hälfte des Path-Framework ausmachen, existieren nicht in Bootstrap. Die Tiles sind die verschiedenfarbigen Buttons wie Contacts, Companies oder Projects in Abbildung 5. Ohne diese verliert das Path-Framework seine gesamte Wirkung. Aus diesem Grund mussten diese ersetzt werden. Dazu wurde der CSS Code aus dem Framework 'metro-bootstrap' zu einem grossen Teil übernommen, einzig die Breite der Tiles wurde verändert. Zuvor hatten die Tiles eine fixe Breite. Da das Path-Framework nun ein GUI für Desktopcomputer und Smartphones erzeugen sollte, musste die Breite der Tiles relativ zu der Breite des Bildschirmes des jeweiligen Gerätes gesetzt werden. Tiles können eine logische Breite von '1' oder '2' haben, wobei ein Tile mit der logischen Breite '2' doppelt so breit ist, wie eines mit der logischen Breite '1'. Für Desktopcomputer nimmt ein Tile mit der logischen Breite '1' nun ein Achtel, und eines mit der logischen Breite '2' ein Viertel der Bildschirmschirmbreite ein (siehe Abbildung 6). Dies ist die einzige Anpassung, die sich auch auf die Desktop-Version des Path-Framework auswirkte. Nun verändert sich die Breite der Tiles auf verschiedenen Endgeräten, je nach Breite des jeweiligen Bildschirms.

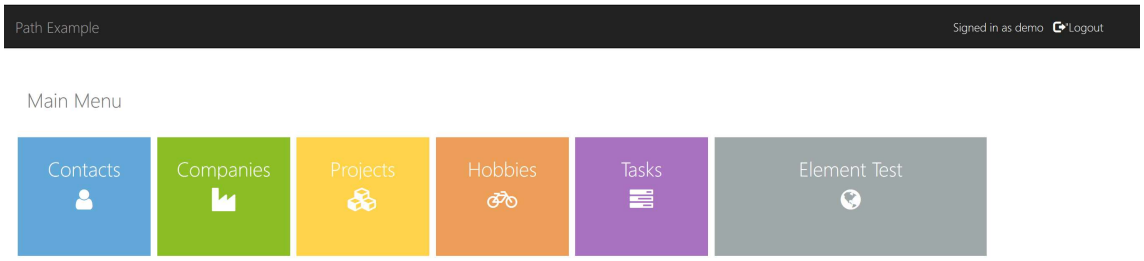


Abbildung 6: Path-Framework inklusive Tiles mit relativer Breite

6.2.2 Entwickeln der mobilen Version

Der erste Schritt auf dem Weg zu der mobilen Version war das Erstellen von Mockups (siehe Abbildung 7). Auf diesen wurde grob festgehalten, wie die mobile Version schlussendlich aussehen soll.

Abbildung 7: Mockups der mobilen Version

Danach musste dieser Plan in die Tat umgesetzt werden, das heisst, der Code des Path-Framework musste erweitert werden.

Das CSS-Framework 'Bootstrap' verändert alle Elemente, die es kennt, zu Responsive-Elementen. Das führt dazu, dass sie sich relativ zu der Bildschirmhöhe und -breite verändern. Somit sahen verschiedene Elemente bereits nach dem Implementieren von Bootstrap ganz okay aus. Trotzdem gab es verschiedene Anpassungen, die noch gemacht werden mussten, bevor das Path-Framework geeignet für mobile Geräte war. In dieser Dokumentation werden die wichtigsten Veränderungen aufgeführt. Zuerst werden Veränderungen auf den Seiten/Pages von Applikationen beschrieben. In einem zweiten Schritt werden Veränderungen an den Formularen aufgezeigt.

Ein Problem war das Breadcrumb, das zur Orientierung auf der Desktop-Version vorhanden ist (siehe Abbildung 8). Dieses nimmt für die mobile Version zu viel Platz ein und wurde aus diesem Grund reduziert. Auf der mobilen Version wird somit nur der letzte Teil des Breadcrumb angezeigt. Auch soll es zentriert erscheinen und somit die Funktion eines Titels auf den jeweiligen Seiten übernehmen.

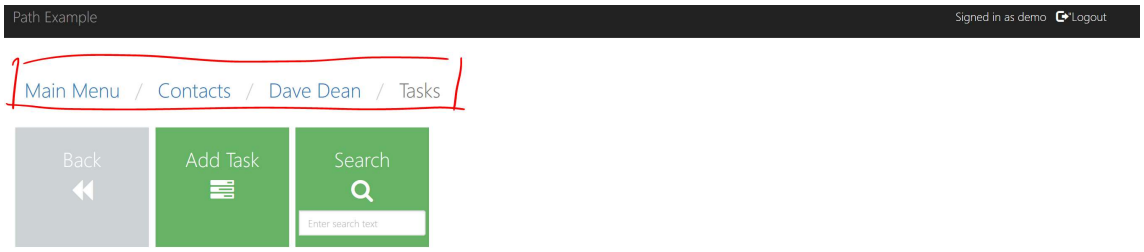


Abbildung 8: Breadcrumb

Wenn der Bildschirm eine gewisse Breite unterschreitet, blendet Bootstrap automatisch den rechten Teil des Headers ('Signed in as' & 'Logout') aus und blendet dafür ein sogenanntes Navigations-Toggle ein. Der Grund für dieses Verhalten ist das Sparen von Platz. Wird nun auf dieses Navigations-Toggle geklickt, soll der versteckte Inhalt angezeigt werden. Dieses Toggle funktionierte zu Beginn nicht. Das Problem konnte jedoch mit kleinen Anpassungen in der path-app.component.html Datei des Path-Framework behoben werden.

Grundsätzlich sollte die Möglichkeit, dass eine logische Breite für Tiles definiert werden kann, auch für die mobile Version beibehalten bleiben. Aus Übersichtlichkeitsgründen wurden jedoch der Back- und der New-Button auf eine logische Grösse von 1 beschränkt. Damit wurde erreicht, dass diese immer oben auf einer Seite angezeigt werden und so nicht zu viel Platz einnehmen.

Eine weitere notwendige Anpassung war die Suchfunktion. Diese ist in der Desktop-Version auch als Tile definiert (siehe Abbildung 8). Da diese Suchfunktion im oberen Bereich einer Seite angezeigt wird, nimmt ein weiteres Tile auf der mobilen Version zu viel Platz ein. Aus diesem Grund wurde die Suchfunktion schlichter und platzsparender gestaltet (siehe Abbildung 9). Zudem sollte die Suchfunktion die ganze Breite des Bildschirms einnehmen, damit sie, auch trotz schlichtem Design, neben den farbigen Tiles nicht untergeht.

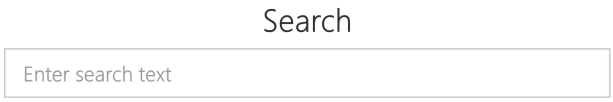


Abbildung 9: Suchfunktion auf der mobilen Version

Letzte grössere Anpassungen, abgesehen von den Formularen, waren für die Listen notwendig. Diese begannen grundsätzlich nicht auf einer neuen Zeile. Auf "normalen" Seiten, wo die Listen über eine Suchfunktion verfügen und keine Inline-Formulare (Formulare, die auf einer Seite integriert sind) eigesetzt werden, fiel dies gar nicht auf, da die Suchfunktion die ganze Bildschirmbreite einnimmt und somit die Liste zwingt, auf einer neuen Zeile zu beginnen. Wurden jedoch Inline-Formulare verwendet, konnte es passieren, dass sich Listenelemente neben andere Elemente zwängten und so das Design der Seite auf einem mobilen Gerät auseinandergerissen wurde. Dies wurde für die mobile Version des Path-Framework angepasst, in dem die Listen in ein HTML-Element gepackt wurden, das als Blockelement definiert ist.

Formulare erscheinen beim Öffnen auf der Desktop-Version vor der gerade geöffneten Seite (siehe Abbildung 10). Zudem kann man sie per Drag-and-Drop auf dem Bildschirm herumschieben. Beide diese Funktionen sind für mobile Endgeräte nicht

praktisch. Einerseits ist der Platz auf einem Smartphone begrenzt, weshalb man das Formular über den ganzen Bildschirm angezeigt haben will. Andererseits will man nicht, dass das Formular verschiebbar ist, da dies über die Bedienung per Touchscreen unabsichtlich passieren kann. Des Weiteren müssen die Buttons (Delete, Cancel und Ok) grösser sein, damit der richtige Knopf über den Touchscreen problemlos gedrückt werden kann. Ebenfalls soll jedes Formular-Feld (Inputfeld) auf einer neuen Zeile beginnen und die Labels (Beschriftung eines Inputfeldes) sollen grösser angezeigt werden.

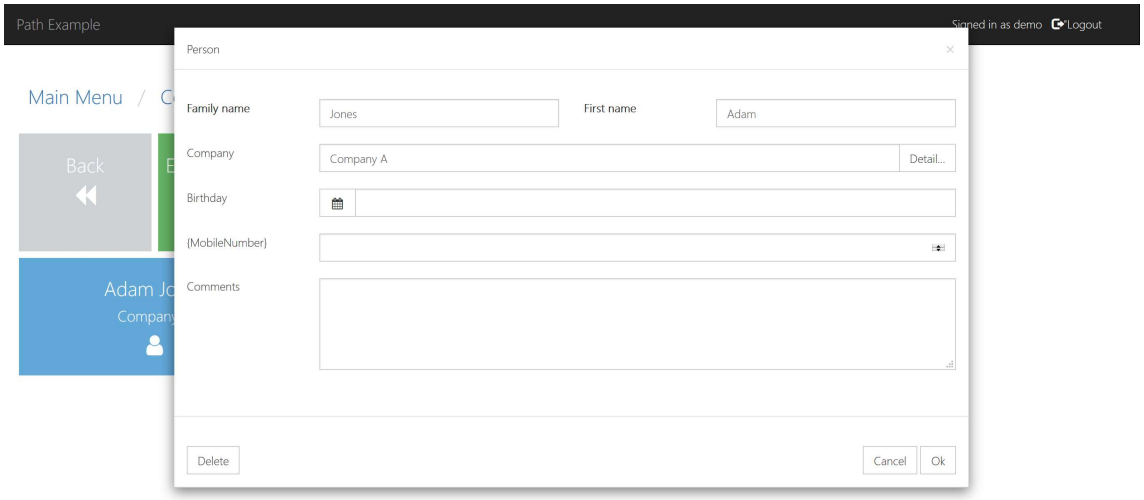


Abbildung 10: Formular auf der Desktop-Version

Ein weiteres Problem war, dass der Footer des Formulars (der Teil, in dem die Buttons sind) immer angezeigt wurde. Problematisch war dies, wenn der Fokus auf einem Inputfeld lag und somit die Tastatur auf dem mobilen Gerät geöffnet wurde. In diesem Fall nahm der Footer einen grossen Teil des restlichen Platzes ein und je nach Grösse des Smartphones, war das eigentliche Formular kaum mehr zu sehen. Aus diesem Grund wird der Footer auf mobilen Geräten entfernt und die Buttons werden unterhalb des eigentlichen Formulars (Body) eingefügt.

Ein Punkt, der auf dem Desktop nicht relevant ist, ist das Anzeigen der richtigen Tastatur je nach Inputfeld. Das heisst, dass bei einem Feld, bei dem Zahlen eingetippt werden müssen, auf einem Smartphone automatisch auch die Tastatur für Zahlen

angezeigt werden muss. Für diesen Punkt mussten glücklicherweise keine weiteren Anpassungen am Path-Framework vorgenommen werden, da dies durch die Typisierung der Inputfelder automatisch gelöst wurde.

Mit dem Abschliessen der Weiterentwicklung des Path-Framework wurde ein wichtiger Meilenstein erreicht. Wie die Smartphone-Version im Vergleich zu der Desktop-Version aussieht, wird im nächsten Abschnitt vorgestellt.

6.2.3 Vergleich Desktop-Version und Smartphone-Version

In diesem Teil wird die Desktop-Version einer Path-Applikation mit der Smartphone-Version verglichen. Es werden die grössten Unterschiede aufgezeigt. Gezeigt wird auch, dass beide Versionen die gleichen Informationen enthalten. Somit besteht kein grosser Mehraufwand mehr, eine Applikation mit dem Path-Framework zu entwickeln, die auf mobilen Betriebssystemen und im Browser laufen soll. Der Vergleich wird mithilfe von Screenshots aus der erstellten Beispielapplikation 'Recipes' visualisiert. Die App 'Recipes' ist eine Rezepte-Applikation, die das Verwalten von Kochrezepten ermöglicht. Eine Live-Version der Desktopversion ist unter 'https://path-recipes.herokuapp.com/' aufgeschaltet. Die App 'Recipes' wurde zudem im Google Play Store veröffentlicht.

6.2.3.1 Hauptmenü

Das Hauptmenü ist die Einstiegsseite einer Path-Applikation. Hier können verschiedene Entitäten angezeigt werden, über welche tiefer in die Applikation hinein navigiert werden kann. Auf der Desktop-Version werden die verschiedenen Buttons nebeneinander aufgelistet. Falls die Buttons aggregiert eine logische Breite von 8 übersteigen, wird eine neue Zeile angefangen. Auf der Smartphone-Version hat man zwei verschiedene Möglichkeiten die Entitäten anzuzeigen: Wählt man eine logische Grösse von '1' pro Button, sind immer zwei Buttons nebeneinander pro Zeile platziert. Die zweite Möglichkeit ist eine logische Grösse von '2' auszuwählen. Somit nimmt jeder Button eine ganze Zeile Platz ein (siehe Abbildung 11). Einen weiteren

Unterschied gibt es in der Kopfzeile, die den Titel der Applikation, den User (Signed in as) und einen Logout-Button enthält. Auf der Smartphone-Version ist nicht genügend Platz, um alle diese drei Elemente nebeneinander anzuzeigen. Aus diesem Grund wird anstatt dem User und dem Logout-Button ein Navigations-Toggle angezeigt (drei Querstriche). Beim Antippen des Toggle erscheinen die beiden Elemente unterhalb der Kopfzeile. Und mit einem zweiten Antippen verschwinden sie wieder.

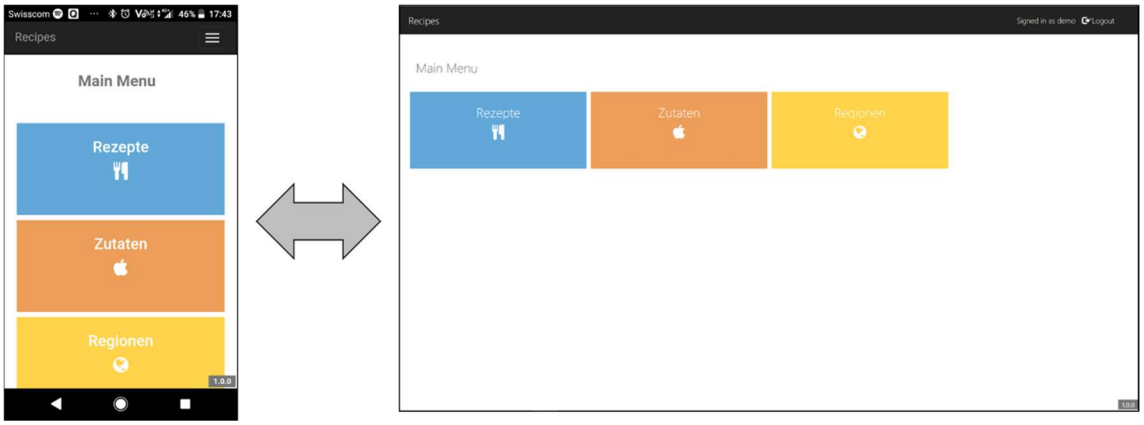


Abbildung 11: Vergleich Smartphone – Desktop: Hauptmenü

6.2.3.2 Rezepte Page

Abbildung 12 zeigt die Seite, die geöffnet wird, wenn auf den Button 'Rezept' geklickt wird. Auf dieser Seite ist eine Liste von Rezepten aufgeführt. Zudem ein Button, mit dem neue Rezepte angelegt werden können und eine Suchfunktion, die das Durchsuchen der Rezeptliste ermöglicht. Listenelemente haben immer eine logische Breite von '2' und nehmen aus diesem Grund auf dem Smartphone je eine Zeile ein. Der Button 'Neues Rezept' kann eine logische Breite von '1' oder '2' annehmen. Ein weiterer Unterschied der beiden Versionen ist die Suchfunktion. Sie hat für das Smartphone ein neues Design erhalten. Das Ziel davon ist Platz einzusparen, damit mehr vom Inhalt der Seite von Anfang an gesehen werden kann.

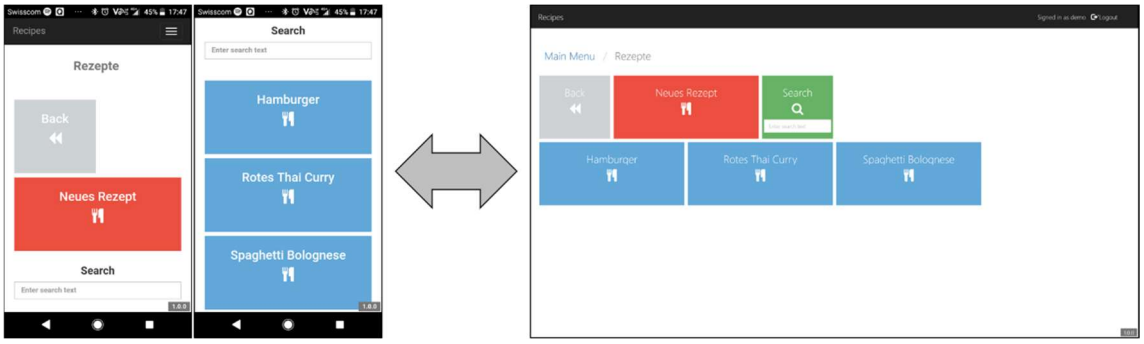


Abbildung 12: Vergleich Smartphone – Desktop: Rezepte Page

6.2.3.3 Hamburger Page

Die Seite, die auf Abbildung 13 gezeigt ist, öffnet sich, wenn auf das Rezept 'Hamburger' geklickt wird. Grundsätzlich ist die Seite gleich aufgebaut, wie die Rezepte Page. Dieses Beispiel soll zeigen, wie die beiden Versionen aussehen, wenn mehr Elemente auf einer Seite enthalten sind.

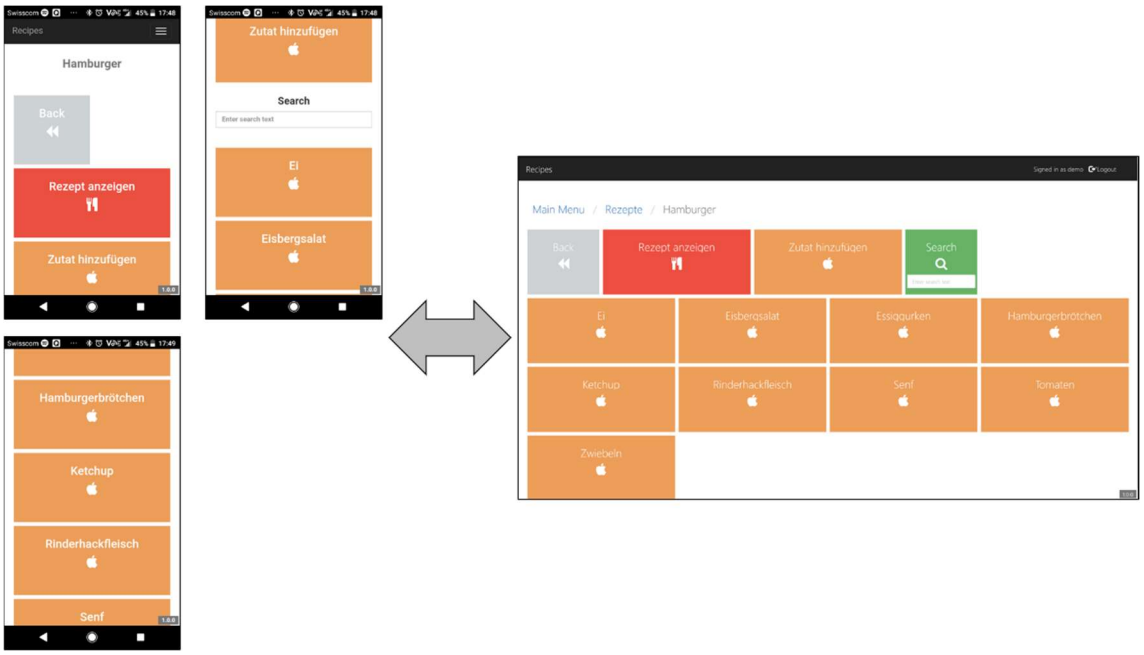


Abbildung 13: Vergleich Smartphone – Desktop: Hamburger Page

6.2.3.4 Formular 'Neues Rezept'

Ein weiterer essentieller Bestandteil des Path-Framework neben den Tiles sind Formulare. Diese wurden für die mobile Version umgestaltet. Der Grund dafür wurde im vorherigen Kapitel erläutert. Abbildung 14 zeigt wie das Formular mit und ohne geöffneter Tastatur auf dem Smartphone aussieht. Zudem sieht man, dass das Formular die gesamte Seite einnimmt, wohingegen es auf dem Desktop über der Seite, von der aus es geöffnet wurde, erscheint.

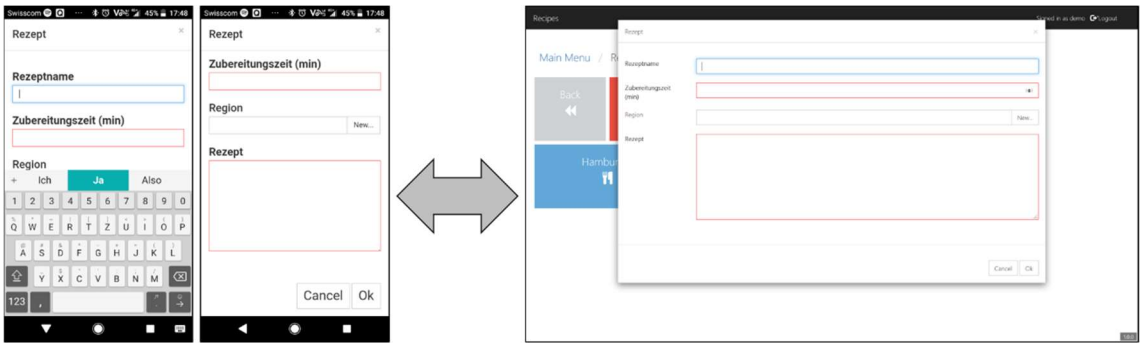


Abbildung 14: Vergleich Smartphone – Desktop: Formular 'Neues Rezept'

Im nächsten Teil wird die Entwicklung der Beispielapplikation 'Recipes' dokumentiert.

6.3 Erstellen einer Beispielapplikation mit dem Path-Framework

Die Grundidee war, eine Beispielapplikation zu erstellen, mit deren Hilfe die mobile Version des Path-Framework validiert werden kann. Dazu musste es eine App sein, die ein Thema behandelt, das von Benutzern aus verschiedenen Altersklassen und beruflichen Hintergründen verstanden wird. Da das Path-Framework auf das Speichern und Anzeigen von Informationen ausgelegt ist, wurde entschieden, eine App zu entwickeln, in der Rezepte angelegt und angezeigt werden können. Die Applikation wurde auf einer bestehenden Path-Applikation aufgebaut. Somit mussten verschiedene Sachen nur noch angepasst und nicht komplett neu erstellt werden. Nach dem Anpassen von verschiedenen Konfigurationen wurde das Graphical User Interface

(GUI) der Applikation erstellt. Hier kam das Path-Framework zum Einsatz. Das heisst, dass die verschiedenen Elemente, die in der Applikation angezeigt werden sollen in einem JSON-Dokument definiert wurden. Aus diesem JSON-Dokument erstellt das Path-Framework dann das GUI der Applikation.

Ein solches JSON-Dokument besteht aus einer Pagelist und einer Formlist (siehe Abbildung 15). In der Pagelist werden die verschiedenen Elemente pro Seite/Page definiert. Und über die Formlist werden die verschiedenen Formulare generiert.

```
{
    "application": {
        "title": "Path Example",
        "formList": [],
        "pageList": []
    }
}
```

Abbildung 15: Leeres JSON fürs Path-Framework-GUI

Abbildung 16 zeigt den Anfang der Pagelist und der Formlist des JSON-Dokumentes für das GUI der 'Recipes' App. Dieses Dokument ist dafür verantwortlich, dass die Applikation so aussieht, wie im Kapitel 6.2.3 gezeigt. Wie solche Seiten und Formulare mit dem Path-Framework angelegt werden können, wird im Readme des Path-Frameworks (GitHub) im Detail erklärt. Grob kann gesagt werden, dass jede Seite und jedes Formular als Liste von verschiedenen Elementen angelegt wird, die dann in dieser Reihenfolge beim Ausführen als HTML-Elemente angezeigt werden.

Pagelist

```
export class GuiModel {

    private _guiModel = {
        "application": {
            "title": "Recipes",
            "pageList": [
                {
                    "id": "mainmenu",
                    "name": "MainMenu",
                    "elementList": [
                        {
                            "type": "button",
                            "name": "Recipes",
                            "icon": "fa-cutlery",
                            "color": "blue",
                            "page": "recipespage",
                            "width": 2
                        },
                        {
                            "type": "button",
                            "name": "Ingredients",
                            "icon": "fa-apple",
                            "color": "carrot",
                            "page": "ingredientspage",
                            "width": 2
                        },
                        {
                            "type": "button",
                            "name": "Regions",
                            "icon": "fa-globe",
                            "color": "yellow",
                            "page": "regionspage",
                            "width": 2
                        }
                    ]
                },
                {
                    "id": "recipespage",
                    "name": "Recipes",
                    "elementList": [
                        {
                            "type": "backbutton",
                        },
                        {
                            "type": "newButton",
                            "name": "NewRecipe",
                            "icon": "fa-cutlery",
                            "color": "red",
                            "width": 2,
                            "form": {
                                "form": "RecipeForm"
                            }
                        },
                        {
                            "type": "list",
                            "name": "RecipeList",
                            "icon": "fa-cutlery",
                            "color": "blue",
                            "search": true,
                            "url": "/recipe",
                            "page": "recipepage",
                        },
                    ]
                },
                {
                    "id": "recipepage",
                    "name": "Recipe",
                    "elementList": [
                        {
                            "type": "backbutton",
                        },
                        {
                            "type": "button",
                            "name": "ShowRecipe",
                            "icon": "fa-cutlery",
                            "color": "red",
                            "width": 2,
                            "form": {
                                "form": "RecipeForm"
                            }
                        },
                        {
                            "type": "button",
                            "name": "AddIngredient",
                            "icon": "fa-apple",
                            "color": "carrot",
                            "width": 2,
                            "form": {
                                "form": "AddIngredientForm"
                            }
```

Formlist

```
            "formList": [
                {
                    "id": "RecipeForm",
                    "title": "Recipe",
                    "url": "/recipe",
                    "formFieldList": [
                        {
                            "id": "recipeName",
                            "type": "text",
                            "name": "RecipeName",
                            "width": 2,
                            "required": true
                        },
                        {
                            "id": "preparationTime",
                            "type": "number",
                            "name": "PreparationtimeInM
                            "width": 2,
                            "required": true
                        },
                        {
                            "id":   "region",
                            "type": "autocomplete",
                            "name": "Region",
                            "wordSearchEnabled": true,
                            "defaultKey": "regionKey",
                            "form": "RegionForm",
                            "url": "/region",
                            "width": 2,
                            "required": false
                        },
                        {
                            "id": "recipe",
                            "type": "text",
                            "name": "Recipe",
                            "width": 2,
                            "height": 8,
                            "maxLength": 50000,
                            "required": true
                        },
```

```
    {
        "id": "IngredientForm",
        "title": "Ingredient",
        "url": "/ingredient",
        "formFieldList": [
            {
                "id": "name",
                "type": "text",
                "name": "IngredientName",
                "width": 2,
                "required": true
            },
            {
                "id": "season",
                "type": "text",
                "name": "Season",
                "width": 2,
                "required": true
            },
            {
                "id": "calories",
                "type": "number",
                "name": "CaloriesPer100g",
                "width": 2,
                "required": true
            },
            {
                "type": "deleteButton",
                "name": "Delete"
            },
            {
                "type": "cancelButton",
                "name": "Cancel"
            },
            {
                "type": "okButton",
                "name": "Ok"
            }
        ]
    },
```

Abbildung 16: JSON von Recipes

Das Back-End einer Path-Applikation kann unabhängig vom Front-End entwickelt werden. Das heisst, dass es bei der Wahl der Technologie für das Back-End keine Einschränkungen gibt. Jedoch wird das Entwickeln des Back-End auch nicht durch das Path-Framework unterstützt. Für die Beispielapplikation Recipes wurde ein einfaches Back-End mit dem Webframework Express.js geschrieben. Als Datenbank wurde PouchDB-Core verwendet. Das Back-End der Applikation ist für die Logik zuständig. Das heisst, dass zum Beispiel bei einem Rezept nur diejenigen Zutaten angezeigt werden, die auch wirklich zu diesem Rezept gehören. Da die Datenbank Änderungen nur in-memory speichert, mussten in der Applikation Testdaten hinterlegt werden. Dazu wurden drei Rezepte mit ihren Zutaten und Herkunftsregionen erfasst (siehe Abbildung 17). Somit wurde die App bei jedem Neustart wieder auf ihre Ausgangslage mit diesen drei Rezepten zurückgesetzt und alle Teilnehmer hatten bei den Usability Tests die gleichen Voraussetzungen.

```
export class TestData {

    public static init() {
        let recipeDatabase = new RecipeDatabase();
        let ingredientDatabase = new IngredientDatabase();
        let regionDatabase = new RegionDatabase();

        let promises = [];
        promises.push(ingredientDatabase.create({name: 'Zwiebeln', season: 'Sommer', calories: '40'}));
        promises.push(ingredientDatabase.create({name: 'Peperoni', season: 'Sommer', calories: '40'}));
        promises.push(ingredientDatabase.create({name: 'Karotten', season: 'Immer', calories: '41'}));
        promises.push(ingredientDatabase.create({name: 'Knoblauch', season: 'Juli bis April', calories: '142'}));
        promises.push(ingredientDatabase.create({name: 'Olivenöl', season: 'Immer', calories: '844'}));
        promises.push(ingredientDatabase.create({name: 'Mais', season: 'August bis Oktober', calories: '365'}));
        promises.push(ingredientDatabase.create({name: 'Rinderhackfleisch', season: '-', calories: '332'}));
        promises.push(ingredientDatabase.create({name: 'Rotwein', season: '-', calories: '85'}));
        promises.push(ingredientDatabase.create({name: 'Gehackte Tomaten', season: '-', calories: '25'}));
        promises.push(ingredientDatabase.create({name: 'Pouletbrust', season: '-', calories: '239'}));
        promises.push(ingredientDatabase.create({name: 'Rote Curry Paste', season: '-', calories: '112'}));
        promises.push(ingredientDatabase.create({name: 'Kokosnussmilch', season: '-', calories: '230'}));
        promises.push(ingredientDatabase.create({name: 'Asiatisches Gemüse', season: '-', calories: '134'}));
        promises.push(ingredientDatabase.create({name: 'Jasminreis', season: '-', calories: '350'}));
        promises.push(ingredientDatabase.create({name: 'Fischsauce', season: '-', calories: '35'}));
        promises.push(ingredientDatabase.create({name: 'Tomaten', season: 'Sommer', calories: '18'}));
        promises.push(ingredientDatabase.create({name: 'Ei', season: '-', calories: '162'}));
        promises.push(ingredientDatabase.create({name: 'Senf', season: '-', calories: '35'}));
        promises.push(ingredientDatabase.create({name: 'Hamburgerbrötchen', season: '-', calories: '237'}));
        promises.push(ingredientDatabase.create({name: 'Essiggurken', season: '-', calories: '11'}));
        promises.push(ingredientDatabase.create({name: 'Eisbergsalat', season: 'Mai bis Oktober', calories: '14'}));
        promises.push(ingredientDatabase.create({name: 'Ketchup', season: '-', calories: '112'}));

        Promise.all(promises).then( onfulfilled: (ingredients) => {
            promises = [];

            promises.push(regionDatabase.create({name: 'Europa'}));
            promises.push(regionDatabase.create({name: 'Asien'}));
            promises.push(regionDatabase.create({name: 'Amerika'}));
```

Abbildung 17: Ausschnitt der Testdaten für die Recipes Applikation

6.4 Wrappen in Apache Cordova

Mit dem Fertigstellen des Back-End war die Recipes Applikation fertig. Sie funktioniert nun im Browser, und wenn das Browserfenster auf die Grösse eines Smartphones reduziert wird, passen sich die Elemente automatisch an. Das Ziel ist jedoch, dass die App auch auf mobilen Betriebssystem, wie Android und iOS, funktioniert und sich verhält, als wäre sie für die jeweiligen Betriebssysteme entwickelt worden (nativ). Um dieses Ziel zu erreichen, musste die Applikation in einen nativen Container gepackt werden. Dies wurde mit dem Mobile Application Development Framework 'Apache Cordova' gemacht. Dieses Framework packt Applikationen, die mit HTML und JavaScript geschrieben wurden, in einen nativen Container, der auf die Gerätefunktionen, wie zum Beispiel die Kamera oder das GPS eines Smartphones, zugreifen kann. Zudem ermöglicht dieser Container das Veröffentlichen von Applikationen auf dem Google Play Store oder im Appstore von Apple.

In nächsten Abschnitt wird das Vorgehen des 'Wrappen' von Recipes in Apache Cordova grob dokumentiert. Im Kapitel '7 Von der Path-Applikation zur mobilen App' (S. 34) wird detailliert beschrieben, wie eine webbasierte Applikation in Apache Cordova gewickelt werden kann.

Um Applikationen in Apache Cordova zu "Wrappen" muss die Cordova CLI (command-line interface), also die Kommandozeile von Apache Cordova, auf dem Rechner installiert sein. Mithilfe von dieser Kommandozeile kann ein neues Apache Cordova Projekt erstellt werden. Diesem Projekt können dann in einem nächsten Schritt diejenigen mobilen Betriebssysteme hinzugefügt werden, auf denen die Applikation funktionieren soll. Nun muss die bestehende Applikation, in diesem Fall Recipes, mit dem Apache Cordova Projekt zusammengeführt werden. Dazu können alle Dateien aus dem Root-Verzeichnis des Recipes Projekts in das Root-Verzeichnis des Cordova Projekts kopiert werden. Einzige Ausnahme bildet hierbei das package.json Dokument. Dieses existiert in beiden Projekten und muss deshalb sorgfältig zusammengefügt werden. Der nächste Schritt ist das Implementieren des Cordova Plugin. Mithilfe von diesem Plugin kann auf die verschiedenen Gerätefunktionen (Kamera, GPS etc.)

zugegriffen werden. Schlussendlich müssen noch einige Konfigurationsänderungen vorgenommen werden. Jetzt kann die App bereits in einem Android oder iOS Emulator gestartet werden (Nnanna, 2017).

Im nächsten Teil wird beschrieben, wie die soeben erstellte hybride Applikation auf dem Google Play Store veröffentlicht werden kann.

6.5 Veröffentlichen der Beispielapplikation

Grundvoraussetzung für das Veröffentlichen von Applikationen ist ein Entwicklerkonto beim gewünschten Anbieter. In diesem Fall wird die Applikation auf dem Google Play Store veröffentlicht. Dies passiert über die Google Play Konsole. Der erste Schritt um eine App auf dem Google Play Store zu veröffentlichen ist das Erstellen eines Android Package (APK) aus den Build-Artefakten der Applikation. Dieses APK ist die Installationsdatei für die App und packt die ganze Applikation in eine einzige Datei. Ein APK kann über die Cordova Kommandozeile erzeugt werden. Nun müssen in der Google Play Konsole verschiedene Informationen hinterlegt werden. Darunter der Verwendungszweck, ein Logo von der App für das Anzeigen im Google Play Store, Screenshots der App und das Android Package an sich. Damit ein Android Package veröffentlicht werden kann, muss es mit einem Schlüssel signiert werden. So ist garantiert, dass allfällige Updates nur von dem Besitzer dieses Schlüssels kommen können und die Nutzer der App vor fremden, möglicherweise schädlichen, Updates geschützt sind. Ein solcher Schlüssel kann mit dem 'Keytool' des Java Development Kit (JDK) erzeugt werden. Das Signieren des APK mit dem erstellten Schlüssel funktioniert zum Beispiel mit dem 'Apksigner' der Android SDK (Software Development Kit) Build Tools. Nachdem das APK erfolgreich hochgeladen und alle benötigten Informationen angegeben wurden, dauert es circa eine Stunde bis die App veröffentlicht ist und über den Google Play Store auf ein Android Gerät heruntergeladen und installiert werden kann. Abbildung 18 zeigt die Seite im Google Play Store von der die App Recipes heruntergeladen werden kann.

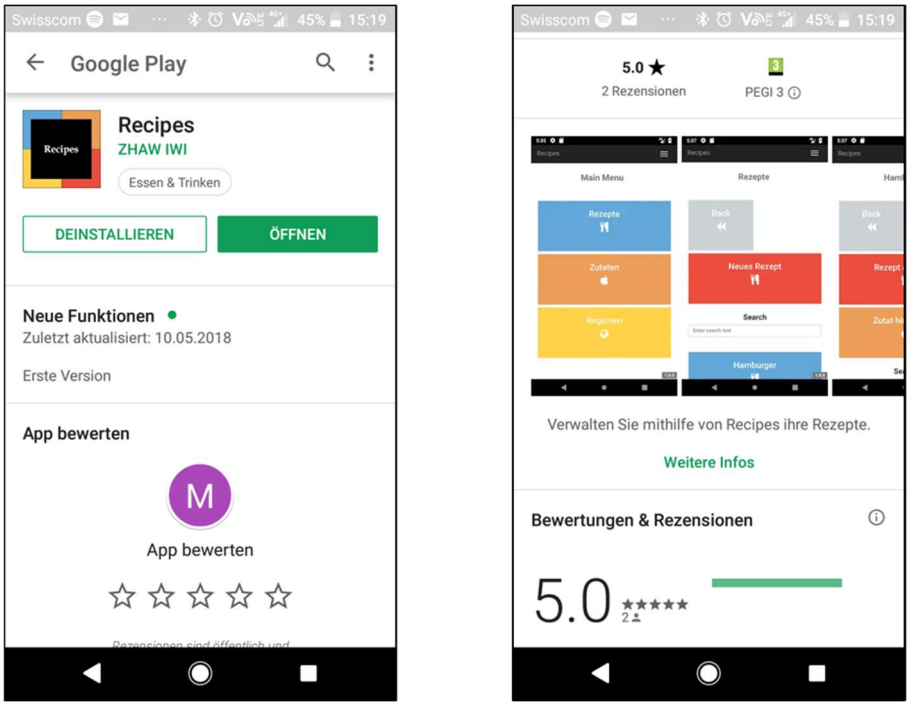


Abbildung 18: Recipes im Google Play Store

7 Von der Path-Applikation zur mobilen App

In dieser Anleitung wird Schritt für Schritt erklärt, wie aus einer Path-Webapplikation eine hybride mobile App erstellt werden kann, die auf jeglichen mobilen Betriebssystemen (Android, iOS etc.) läuft. Um diese Anleitung anwenden zu können, müssen folgende Voraussetzungen gegeben sein:

- Path-Applikation (Front-End)
- Back-End
- Installation der Cordova CLI
- Installation der Angular CLI

Diese Anleitung ist auch im Path Wiki auf Englisch aufgeschaltet (siehe Anhang 12).

1. Cordova Create:

Als erstes muss ein Apache Cordova Projekt erstellt werden. Dies wird über die Cordova CLI gemacht. Dafür muss zuerst in das gewünschte Verzeichnis navigiert und dort der Kommandozeilenbefehl '*cordova create [path] [id [name [config]]] [title]*' eingegeben werden (siehe Abbildung 19). Hierbei stellt [path] den Namen des Ordners dar, in dem das Projekt erstellt werden soll. [id [name [config]]] ist ein reverse domain-style Identifikator. Und [title] ist der gewünschte Titel des Projektes. Ein Beispiel sieht folgendermassen aus:

```
C:\pathProject\Beispielapplikationen>cordova create recipesCordova com.herokuapp.pathRecipes RecipesCordova
Creating a new cordova project.

C:\pathProject\Beispielapplikationen>
```

Abbildung 19: Erstellen eines Apache Cordova Projekts mit der Cordova CLI

Weitere Informationen zu Cordova Create können in der Dokumentation von Apache Cordova gefunden werden.

2. Cordova Platform Add:

Der zweite Schritt ist das Hinzufügen der gewünschten Plattformen für das Cordova Projekt. Dazu muss in die Ordnerstruktur des Projektes navigiert werden. Dort können

mit dem Befehl '*cordova platform add [platform]*' die gewünschten Plattformen hinzugefügt werden (siehe Abbildung 20). [platform] steht hierbei für das gewünschte Betriebssystem, also zum Beispiel Android oder iOS.

```
C:\pathProject\Beispielapplikationen\recipesCordova>cordova platform add android

C:\pathProject\Beispielapplikationen\recipesCordova>cordova platform add browser
```

Abbildung 20: Hinzufügen von Plattformen zu einem Cordova Projekt

Wichtig zu erwähnen ist hierbei, dass für das Bauen von Applikationen für ein bestimmtes Betriebssystem das jeweilige Software Development Kit (SDK) auf dem Computer installiert sein muss. Die einzige Ausnahme hier ist die Plattform 'Browser'. Das SDK für iOS ist nur für Mac Computer verfügbar. Das führt dazu, dass grundsätzlich von einem Windows Rechner aus keine Apps für iOS entwickelt werden können.

3. Cordova Requirements:

Abbildung 21 zeigt, wie mit dem Befehl '*cordova requirements*' über die Cordova CLI abgeklärt werden kann, ob die Voraussetzungen für die jeweiligen Plattformen erreicht sind.

```
C:\pathProject\Beispielapplikationen\recipesCordova>cordova requirements
Android Studio project detected

Requirements check results for android:
Java JDK: installed 1.8.0
Android SDK: installed true
Android target: installed android-27,android-26
Gradle: installed C:\Program Files\Android\Android Studio\gradle\gradle-4.4\bin\gradle
```

Abbildung 21: Abklärung der Voraussetzungen für Android

Es kann sein, dass folgende Fehlermeldung angezeigt wird (siehe Abbildung 22). Falls dies der Fall sein sollte, ist in diesem Abschnitt beschrieben, wie dieser Fehler für Windows behoben werden kann. Falls dieses Problem nicht besteht, kann der Rest dieses Abschnitts übersprungen werden.

```
C:\pathProject\Beispielapplikationen\recipesCordova>cordova requirements
Android Studio project detected

Requirements check results for android:
Java JDK: installed 1.8.0
Android SDK: installed true
Android target: not installed
Please install Android target / API level: "android-26".

Hint: Open the SDK manager by running: "C:\Users\Manu\AppData\Local\Android\sdk\tools\android.bat"
You will require:
1. "SDK Platform" for API level android-26
2. "Android SDK Platform-tools (latest)
3. "Android SDK Build-tools" (latest)
Gradle: installed C:\Program Files\Android\Android Studio\gradle\gradle-4.4\bin\gradle
(node:15924) UnhandledPromiseRejectionWarning: CordovaError: Some of requirements check failed
```

Abbildung 22: Fehlermeldung betreffend Voraussetzungen für Android

Schritt für Schritt Lösung

1. Öffnen von Android Studio
2. Configure → SDK Manager
3. Anschliessend öffnet sich ein Dialogfeld.
4. Hier muss das Häkchen bei 'Android 8.0 (Oreo), API Level 26' gesetzt werden
5. Anschliessend auf Apply klicken
6. Nun wird das API Level 26 installiert
7. Nach Abschluss der Installation nochmals 'cordova requirements' über die Konsole ausführen – die Fehlermeldung sollte jetzt nicht mehr erscheinen.

4. Zusammenführen des Angular und des Cordova Projektes:

Der nächste Schritt ist das bestehende Angular Projekt mit dem Cordova Projekt zusammenzuführen. Hierzu werden alle Ordner und Files, **ausser package.json**, aus dem Angular Root-Directory in das Cordova Root-Directory kopiert (siehe Abbildung 23).

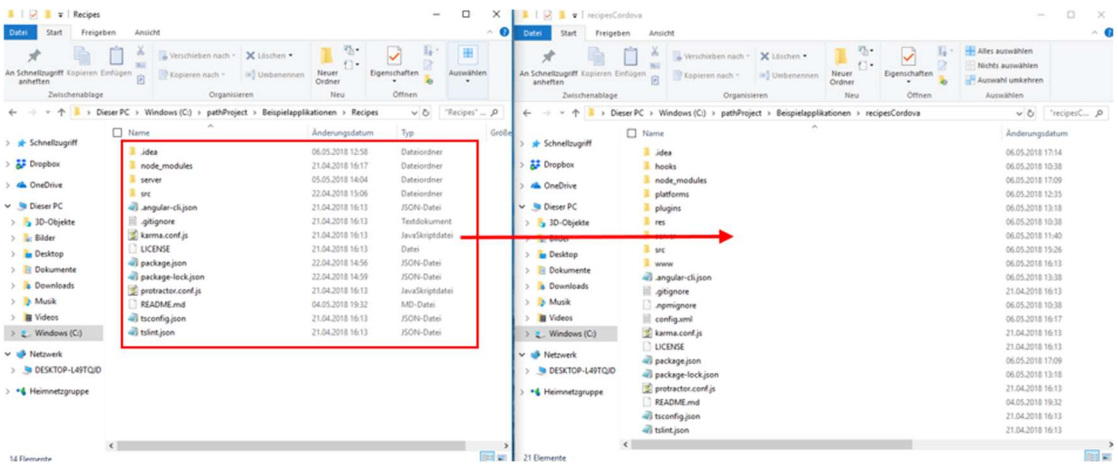


Abbildung 23: Kopieren des Angular Projektes in das Cordova Projekt

Hinweis: Der Ordner node_modules muss nicht kopiert werden, da dieser mit npm install im Cordova Projekt zu einem späteren Zeitpunkt installiert werden kann.

5. Zusammenführen der package.json Files:

Nun werden die beiden package.json Dateien zu einer zusammengeführt. Dazu werden beide geöffnet und vorsichtig zusammengeführt. Die daraus resultierende package.json Datei sollte in etwa aussehen, wie die in Abbildung 24.

package.json

```
{
  "name": "com.herokuapp.pathrecipes",
  "displayName": "RecipesCordova",
  "version": "1.0.0",
  "description": "Recipes Sample Application, which uses the Path-Framework and is wrapped in Apache Cordova to act as a
  "main": "index.js",
  "keywords": [
    "Recipes",
    "GUI",
    "JSON",
    "REST",
    "Rapid Application Development",
    "Rapid Application Manufacturing"
  ],
  "homepage": "https://github.com/zhaw-weberm16/Recipes/",
  "bugs": {
    "url": "https://github.com/zhaw-weberm16/Recipes/issues"
  },
  "author": {
    "name": "Manuel Weber"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/zhaw-weberm16/Recipes.git"
  },
  "engines": {
    "node": "7.10.1",
    "npm": "4.0.5"
  },
  "scripts": {
    "ng": "ng",
    "start": "tsc --project server/tsconfig.json && concurrently \"ng serve\" \"nodemon server/server.js\"",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "heroku-postbuild": "tsc --project server/tsconfig.json && ng build"
  },
  "license": "Apache-2.0",
  "dependencies": {
    "body-parser": "1.15.2",
    "cordova-android": "^7.0.0",
    "cordova-browser": "^5.0.3",
    "cordova-plugin-whitelist": "^1.3.3",
    "express": "4.15.0",
    "path-framework-weberm16": "1.1.5",
    "pouchdb-adapter-memory": "6.3.4",
    "pouchdb-core": "6.3.4"
  },
  "devDependencies": {
    "@angular/cli": "1.6.5",
    "@angular/compiler-cli": "4.2.5",
    "@types/jasmine": "2.5.38",
    "@types/node": "6.0.60",
    "codelyzer": "2.0.0",
    "concurrently": "2.2.0",
    "jasmine-core": "2.5.2",
    "jasmine-spec-reporter": "3.2.0",
    "karma": "1.4.1",
    "karma-chrome-launcher": "2.0.0",
    "karma-cli": "1.0.1",
    "karma-jasmine": "1.1.0",
    "karma-jasmine-html-reporter": "0.2.2",
    "karma-coverage-istanbul-reporter": "0.2.0",
    "nodemon": "1.11.0",
    "protractor": "5.1.0",
    "ts-node": "2.0.0",
    "tslint": "4.4.2",
    "typescript": "2.1.5"
  },
  "cordova": {
    "plugins": {
      "cordova-plugin-whitelist": {}
    },
    "platforms": [
      "android",
      "browser"
    ]
  }
}
```

Abbildung 24: Package.json Datei nach dem Zusammenführen

6. Npm Install:

Nun, da die package.json Dateien zusammengeführt wurden, kann im Cordova Verzeichnis der Kommandobefehl '*npm install*' ausgeführt werden. Somit werden alle Dependencies des Projektes installiert und im Ordner node_modules abgelegt.

7. Ng Serve:

Nach dem Abschliessen der obenstehenden Punkte wurde das Angular Projekt erfolgreich zu einer mobilen App konvertiert. Zum Überprüfen, ob alles geklappt hat, kann '*ng serve*' für einen Entwicklungsserver ausgeführt werden. Somit wird die Applikation unter der URL 'http://localhost:4200/' geladen. Jegliche Änderungen, die nun an der Applikation vorgenommen werden, führen dazu, dass der Browser neu lädt und die vorgenommenen Änderungen sogleich anzeigt.

8. Implementierung des Cordova Plugin:

Um das Cordova Plugin zu implementieren muss die folgende Zeile Code in die index.html Datei des Projektes kopiert werden:

```
<script type="text/javascript" src="cordova.js"></script>
```

Hinweis: Die Datei cordova.js muss nicht existieren, sie wird beim Build-Prozess von Cordova erzeugt.

Von nun an kann auf die verschiedenen Plugins von Cordova zugegriffen werden. Über diese Plugins kann zum Beispiel auf die Kamera oder den Standort von einem Gerät zugegriffen werden.

9. Build und Run der Applikation:

Damit der Build und der Run funktionieren, müssen noch ein paar kleine Anpassungen vorgenommen werden:

1. Zuerst muss der <base href="/"> Tag in der index.html Datei auf <base href="./"> angepasst werden. Dies ermöglicht es Cordova, Dateien in einem Verzeichnispfad zu erreichen.
2. Angular erstellt einen 'dist' Ordner beim Build mit den Output Files. Cordova verwendet hingegen den 'www' Ordner. Aus diesem Grund muss Angular so angepasst werden, dass es das Projekt in den www Ordner "hineinbaut". Dies wird

erreicht indem in der .angular-cli.json Datei der Wert von 'outDir' von 'dist' auf 'www' angepasst wird.

3. Im www Ordner des Cordova Projektes sind bis jetzt noch Dokumente von der Standard Cordova Applikation. Aus diesem Grund kann der www Ordner gelöscht werden.
4. Anschliessend kann über die Kommandozeile 'ng build' ausgeführt werden. Somit wird ein neuer www Ordner mit den Output-Files der Angular Applikation erstellt.

10. Testen mit einem Android Emulator:

Der einfachste Weg zu testen, ob die Applikation auf einem Betriebssystem funktioniert, ist mithilfe eines Emulators. Mit der Installation von Android Studio wird standardmässig auch ein Android Emulator mitinstalliert. Um mit dem Android Emulator ein virtuelles Gerät zu erstellen, muss der AVD (Android Virtual Device) Manager über Android Studio gestartet werden (siehe Abbildung 25).

Abbildung 25: AVD Manager in Android Studio

Hier kann nun ein virtuelles Gerät erstellt werden. Sobald ein Android Emulator erstellt wurde, kann dieser gestartet werden (über den AVD Manager). Nun kann das Cordova Projekt über die Kommandozeile mit dem Befehl 'cordova run android' gestartet werden. Falls kein physisches Gerät gefunden wird, startet die Applikation standardmässig im Emulator. Somit wurde das Angular Projekt erfolgreich in einen nativen Container gepackt.

Quelle: Nnanna, 2017

Diese Anleitung wird zusätzlich auf Englisch im Wiki vom Path-Framework auf GitHub veröffentlicht (Anhang 12).

8 Usability Tests

Für das Validieren der Forschungsfrage wurden Usability Tests mit verschiedenen Testpersonen durchgeführt. Das Ziel dieser Usability Tests ist die Benutzerfreundlichkeit und die Verständlichkeit einer Path-Applikation abzuklären. Dies wurde mithilfe der Beispielapplikation 'Recipes' getestet.

8.1 Methodik

Der Usability Test wurde mit 10 Personen im Alter von 20 bis 60 Jahren durchgeführt. Die Tests sind so abgelaufen, dass die Testperson gebeten wurde, sich die App über den Play Store herunterzuladen, vorausgesetzt sie besitzt ein Android Smartphone. Als erstes verschaffte sich die Testperson einen Überblick über die App und schilderte ihren ersten Eindruck. Danach musste die Testperson acht verschiedene Szenarien durcharbeiten. Dabei wurde sie beobachtet und gebeten laut zu denken. Alle Beobachtungen und Gedanken wurden notiert. Nach jedem Szenario wurde die Testperson gefragt, was positiv und was negativ aufgefallen war. Nach dem Durcharbeiten der Szenarien wurde nochmals eine Rückmeldung über den gesamten Test eingeholt. Zudem wurde nach Verbesserungsvorschlägen und Erweiterungspotentialen für die Path-Applikation gefragt. Als letztes musste sich die Testperson in Bezug auf ihre Technologiekenntnisse auf einer Skala von 1 bis 5 selbst einschätzen und die Benutzerfreundlichkeit und die Verständlichkeit der App je auf einer Skala von 1 bis 10 bewerten. Ein solcher Test dauerte im Durchschnitt circa 30 Minuten. Der Fragebogen für den Usability Test und die Beobachtungsbögen von den einzelnen Teilnehmenden sind im Anhang aufzufinden (Anhang 1 – 11).

8.2 Ergebnisse und Validation

8.2.1 Testpersonen

Von den Testpersonen waren vier weiblich und sechs männlich. Fünf Personen waren unter 25, drei zwischen 25 und 40, eine zwischen 41 und 55, sowie eine über 55. Die Usability Tests wurden direkt auf den Smartphones der Testpersonen durchgeführt, um aufzuzeigen, dass die App auf verschiedenen Geräten funktioniert. Somit wurden acht Tests auf sechs verschiedenen Samsung Geräten und zwei Tests auf einem Sony Xperia Z3 Compact durchgeführt. Es wurde versucht Personen mit möglichst vielfältigen beruflichen Hintergründen für die Tests auszuwählen. Unter den Testpersonen befanden sich Studenten aus verschiedenen Studienrichtungen, zwei Hauswarte, eine Lehrerin, ein Richter, ein Automechaniker, eine Person aus dem Bereich Technik und eine Schulleiterin. Zudem wurde darauf geschaut, dass die Testpersonen verschieden gute Technologiekenntnisse mitbringen. Wie sich die Testpersonen in Bezug auf ihre Technologiekenntnisse auf einer Skala von 1 bis 5 eingeschätzt haben, ist Abbildung 26 zu entnehmen.

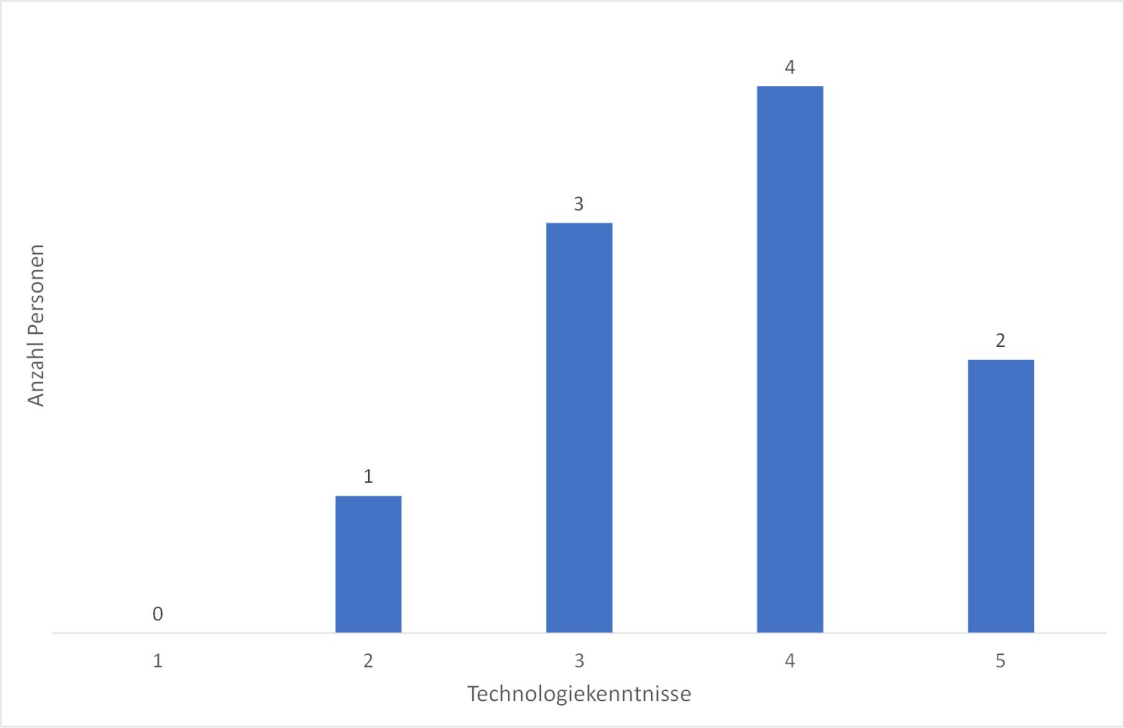


Abbildung 26: Selbsteinschätzung Technologiekenntnisse

8.2.2 Allgemeiner Eindruck

Grundsätzlich ist der allgemeine Eindruck der App sehr positiv ausgefallen. Sie wurde von den Testpersonen als selbsterklärend, einfach handzuhaben, übersichtlich und handlich beschrieben. Es sei intuitiv klar was wo gefunden werden kann und wie die App aufgebaut ist. Man finde sich gut zurecht und die einzelnen Elemente seien gut lesbar. Weiter wurde gesagt, dass die App sehr übersichtlich ist. Dies weil sie konsistent aufgebaut ist und die Elemente immer wieder gleich aussehen. Die App wird als sehr einfach und aus diesem Grund als problemlos bedienbar und handlich wahrgenommen. Somit haben sich alle Testpersonen nach kurzer Zeit gut zurechtgefunden. Negative Bemerkungen fielen rar aus. Gesagt wurde, dass die App klobig und bausatzmässig aussehe. Und dass es auf den einzelnen Seiten nicht klar sei, dass man nach unten scrollen kann. Die Farben und die Icons der Applikation wurden als positiv eingestuft. Sie wurden als gute Orientierungshilfe gesehen. Auch dass man über verschiedene Wege an Informationen gelangen kann, wurde als positive Eigenschaft erwähnt und die Suchfunktion sei sehr praktisch.

8.2.3 Szenarien

Die Testpersonen mussten während dem Usability Test acht verschiedene Szenarien durcharbeiten. In diesen ging es darum bestehende Elemente zu löschen, neue Elemente anzulegen oder bestimmte Informationen aus der App auszulesen. In Szenario 1 musste eine Zutat aus einem bestimmten Rezept herausgelöscht werden. Szenario 2 und 3 befassten sich mit dem Hinzufügen von Zutaten zu bestehenden Rezepten. In Szenario 4 bis 7 mussten verschiedene Informationen in der App gefunden und in Szenario 8 ein neues Rezept angelegt werden. Grundsätzlich konnten alle Szenarien von allen Testpersonen problemlos durchgearbeitet werden und das Vorgehen wurde als selbsterklärend beschrieben. Vor allem das Herauslesen von Informationen war für alle Personen selbsterklärend und auch für diejenigen Nutzer mit schlechteren Technologiekenntnissen keine Herausforderung. Probleme gab es hingegen beim Hinzufügen von einer Zutat zu einem Rezept. Hier sagten 80% der Testpersonen, dass diese Funktion intuitiv nicht klar sei. Das Problem entstand, weil nicht klar ist, dass

unterschieden werden muss, ob eine Zutat bereits existiert oder nicht. Abbildung 27 zeigt wie eine Zutat, die noch nicht existiert, hinzugefügt werden muss. Jedoch waren alle zehn Testpersonen, nachdem sie ein bisschen herumprobiert hatten, in der Lage eine neue Zutat anzulegen und diese dem Rezept hinzuzufügen. Zwei Personen verstanden das Hinzufügen einer neuen Zutat sogar auf Anhieb. Ein weiteres Problem gab es beim Erstellen eines neuen Rezeptes. Das Rezept konnten alle Testpersonen problemlos anlegen, aber nur 7 Personen dachten daran nach dem Erstellen des Rezeptes die benötigten Zutaten hinzuzufügen. Hier war die Herausforderung, dass die Zutaten nicht vom Formular 'Neues Rezept hinzufügen' aus hinzugefügt werden können, sondern erst nach dem Erstellen des Rezeptes über ein separates Formular. Auch die Personen, die an das Hinzufügen von Zutaten gedacht hatten, waren der Meinung, dass dies leicht vergessen werden könne. Hier wurde vorgeschlagen mindestens eine Messagebox anzuzeigen, die nach dem Erstellen des Rezeptes darauf hinweist, dass noch die entsprechenden Zutaten hinzugefügt werden müssen.

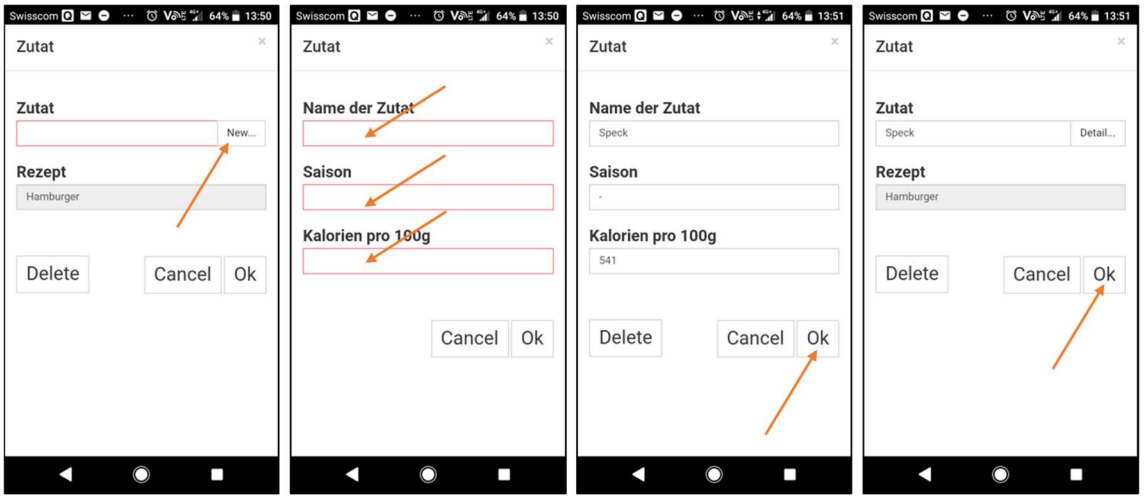


Abbildung 27: Hinzufügen einer Zutat, die noch nicht existiert

8.2.4 Benutzerfreundlichkeit und Verständlichkeit

Alle Testpersonen wurden nach dem Beenden des Usability Tests gebeten die Benutzerfreundlichkeit und die Verständlichkeit von Recipes auf einer Skala von 1 bis 10 zu bewerten, wobei 1 die schlechteste und 10 die beste Bewertung darstellte. Unter der Benutzerfreundlichkeit wird verstanden, wie ansprechend die App ist, wie angenehm sie zu bedienen ist und wie wohl man sich beim Benutzen fühlt. Verständlichkeit meint, wie selbsterklärend und intuitiv bedienbar die Applikation ist. Beide Punkte erzielten einen sehr guten Wert. Die Benutzerfreundlichkeit wurde im Durchschnitt mit 8.3 und die Verständlichkeit mit 8.2 bewertet. Da es bei beiden Bewertungspunkten keine extremen Ausreisser gab, die das arithmetische Mittel verfälschen würden, wurde dieses für die Berechnung des Durchschnittes verwendet. Abbildung 28 und 29 zeigen wie die einzelnen Bewertungen verteilt sind. Aufgrund von diesen Erkenntnissen kann gesagt werden, dass es möglich ist mit dem Path-Framework hybride Applikationen zu entwickeln, die sowohl auf dem Desktop und dem Smartphone funktionieren und zudem benutzerfreundlich und verständlich für den Endnutzer sind.

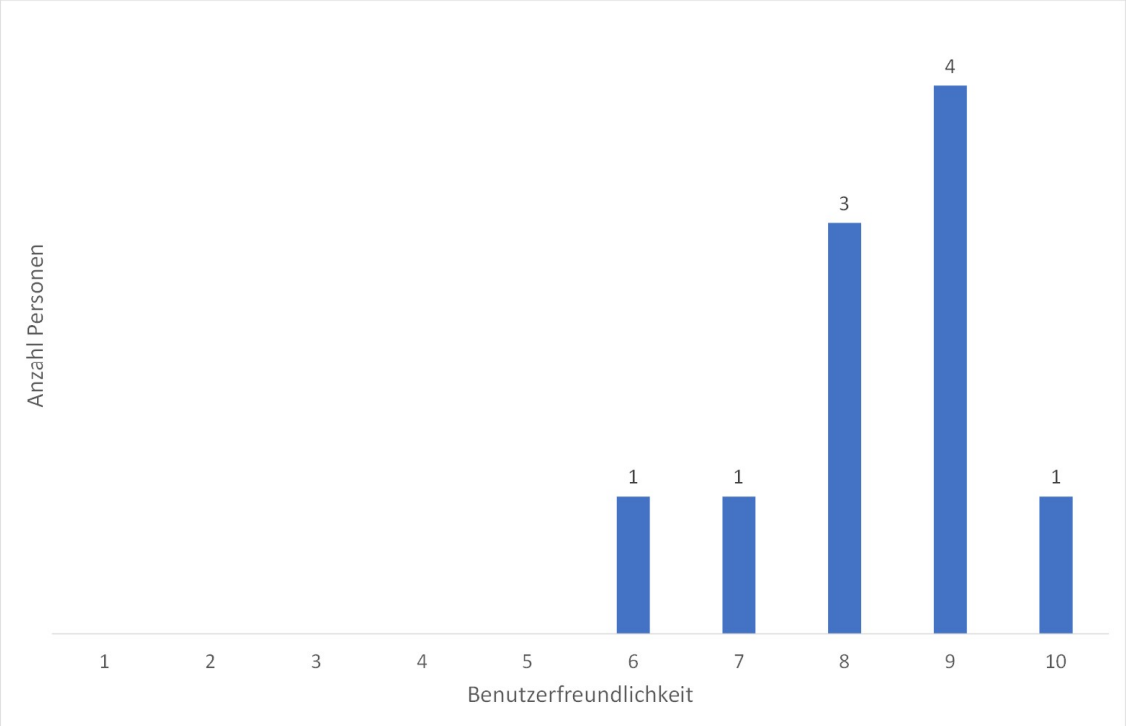


Abbildung 28: Verteilung Benutzerfreundlichkeit

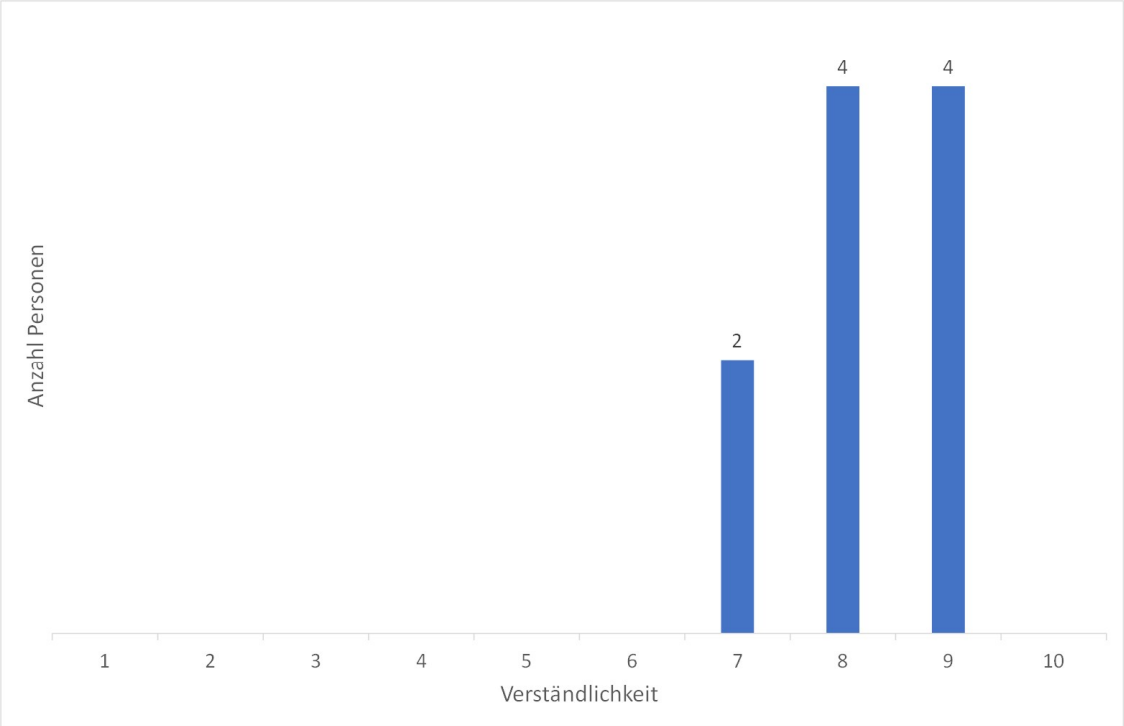


Abbildung 29: Verteilung der Verständlichkeit

8.2.5 Verbesserungs- und Erweiterungsvorschläge

Auch wenn die Path-Beispielapplikation sehr positiv bewertet worden ist, kamen verschiedene Punkte auf, die verbessert werden können. Zusätzlich sind während den Tests auch einige wenige Erweiterungsideen entstanden. Ein Punkt, der nahezu von allen Testpersonen bemängelt wurde ist, dass der native Backbutton eines Android-Gerätes aus der App hinaus navigiert. Hier wurde gewünscht, dass dieser einen Schritt zurücknavigiert, sofern man sich nicht im Hauptmenü der App befindet. Und dass ein Hinweis erscheint, falls man den Backbutton im Hauptmenü drückt, dass somit die App verlassen wird. Weiter sorgte der Backbutton der Path-Applikation für Kritik. Dies weil er zu gross sei und somit zu viel Platz des Smartphone-Bildschirms einnehme. Vorgeschlagen wurde, dass dieser neben dem Titel (Breadcrumb) der jeweiligen Seiten platziert werden solle. Ein weiterer Punkt betraf die Suchfunktion. Sie wurde als sehr praktisch und hilfreich beschrieben, jedoch wurde das schlichte Design bemängelt. Drei Personen gaben an, die Suchfunktion gar nicht bemerkt zu haben. Auch Personen, welche die Suchfunktion bemerkten, waren der Meinung, dass diese neben den farbigen Tiles untergehe. Aus diesem Grund solle sie auffälliger gestaltet

werden. Ein Beispiel wäre das Hinterlegen der momentanen Suchfunktion mit einer Farbe. Auch die Benennung der Suchfunktion wurde vermehrt angesprochen. Unter einer Suchfunktion versteht der Endnutzer eine Funktion, welche die ganze Applikation nach bestimmten Schlagwörtern durchsucht. Diese sucht der Nutzer typischerweise ganz oben in einer App. Bei der Suchfunktion des Path-Frameworks handelt es sich jedoch um einen Funktion, die eine bestimmte Liste durchsucht. Aus diesem Grund macht die Platzierung direkt oberhalb der Liste Sinn, sie müsse jedoch anders beschriftet sein. Vorgeschlagen wurde als Beispiel für das Durchsuchen der Liste von Zutaten 'Zutaten durchsuchen' oder einfach nur 'Durchsuchen'. Ein weiterer Vorschlag war, dass die jeweiligen Suchfunktionen im JSON-Dokument benennt werden können. Bei den Formularen wurde bemängelt, dass diese 'alt' und nach Programmierumgebung aussähen. Somit müsste das Design der Formulare für Smartphones angepasst werden. Am besten so, dass sie ähnlich aussehen wie Standardformulare in anderen Apps. Hierbei wurde vor allem, aber nicht nur, das Design der Buttons (Ok, Cancel & Delete) angesprochen. Diese seien zu gross und es wäre von Vorteil, wenn sie treffender angeschrieben wären. Als Beispiel für eine treffendere Bezeichnung für den Ok-Button wurde 'Änderungen speichern' genannt. Weiter wurde der Button, der zurück ins Hauptmenü navigiert, nicht wahrgenommen und sogar vermisst. Momentan ist der Titel der App (oben links) gleichzeitig der Button, der zurück ins Hauptmenü navigiert. Dies ist jedoch nur auf Webseiten üblich. Aus diesem Grund müsste dies in der mobilen Version ein typisches Home-Button-Logo sein. Nebst den bisher genannten Verbesserungsvorschlägen wurden weitere kleinere Änderungen vorgeschlagen. Ein Punkt hier ist, dass die Inputfelder beim Öffnen eines Formulars nicht direkt Fokus haben sollen. Der Grund hierfür ist, dass wenn ein Inputfeld Fokus hat, die Tastatur sogleich geöffnet wird und die Hälfte des Bildschirms einnimmt. Weiter soll für Smartphones die Hover-Funktion (kommt zum Zug, wenn ein Element lange angetippt wird) entfernt werden, da diese für Verwirrung sorgt. Weiter wurde gesagt, dass es schön wäre, wenn man den Listen Titel geben könnte. Somit könnte die Liste von Zutaten eines Rezeptes den Titel 'Zutaten für das Rezept (XY)' tragen. Der letzte Punkt, der erwähnt wurde war, dass die bestehenden Rezepte beim Öffnen nicht direkt bearbeitbar sein sollen. Einerseits damit nicht ausversehen etwas geändert wird und andererseits, dass die Tastatur nicht automatisch von Anfang an angezeigt wird. Als

Erweiterungspotential wurde die Möglichkeit genannt, N zu M Beziehungen direkt in einem Formular, mithilfe von Buttons in den Formularen, abbilden zu können. Weiter wurde die Möglichkeit, den Buttons verschiedene Formen für eine weitere optische Differenzierung zu geben, oder die Formulare farbig hinterlegen zu können, genannt. Zu all den in diesem Abschnitt erwähnten Punkten muss gesagt sein, dass von den Testpersonen verdeutlicht wurde, dass es sich hierbei um Kritik 'auf hohem Niveau' handle.

8.2.6 Bugs

Während den Usability Tests sind drei Bugs aufgetaucht. Wichtig zu erwähnen ist hierbei, dass die Bugs nur in der mobilen App vorkommen. In der Webversion derselben App bestehen diese nicht. Alle Bugs ähneln sich in einem: Sie führen zum Absturz der App, wenn auf bestimmte Sachen getippt wird. Das Erste, das die App zum Absturz bringt, ist das Auswählen eines Elementes aus einer Autocomplete-Liste. Der zweite Punkt ist das Verwenden des Homebuttons oben links. Und der dritte Punkt, der einen Absturz verursacht, ist das Antippen des Logouts. All diese Punkte führen zu der gleichen generischen Fehlermeldung (siehe Abbildung 30).

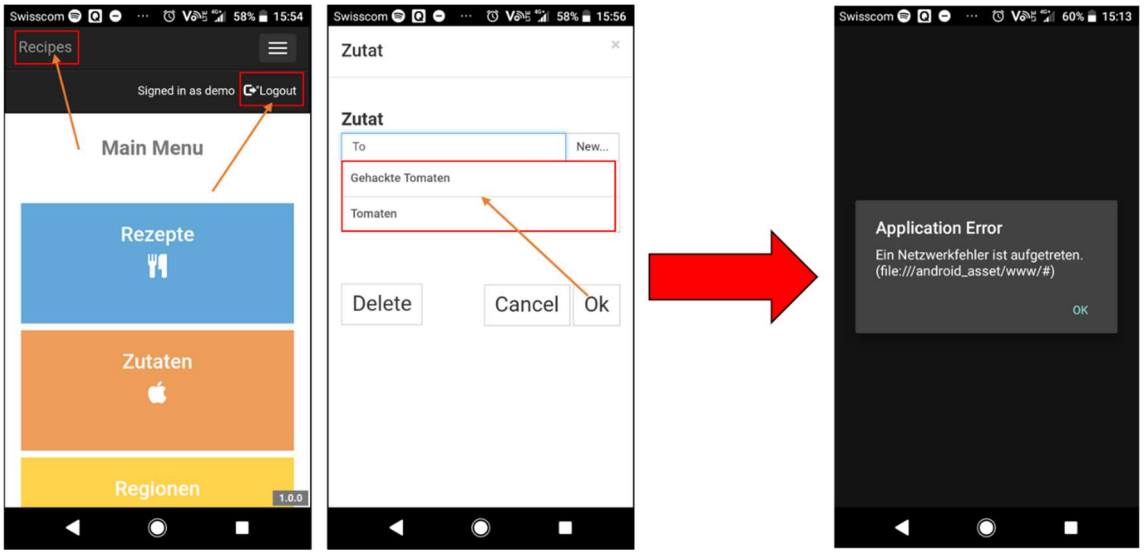


Abbildung 30: Absturz der App

9 Konklusion

In der Konklusion wird die Forschungsfrage nochmals aufgegriffen. Es wird ein kurzer Ausblick betreffend RAD-Tools gegeben und abgeschlossen wird das Thema mit einer kurzen persönlichen Reflexion.

Mit dem Abschluss dieser Bachelorarbeit können nun hybride Applikationen mit dem Path-Framework entwickelt werden. Der Entwicklungsaufwand wird durch das Einsetzen dieses Tools massgeblich reduziert und die erstellten Apps sind benutzerfreundlich und verständlich für den Endnutzer. Somit kann die Forschungsfrage dieser Arbeit mit 'Ja' beantwortet werden: Ja, es ist möglich mit dem Path-Framework hybride Applikationen zu entwickeln, die auf dem Desktop und dem Smartphone funktionieren und zudem für den Endnutzer benutzerfreundlich und verständlich sind. Es kann davon ausgegangen werden, dass die Nachfrage nach RAD-Tools in der Zukunft mindestens gleich hoch bleiben wird, denn die Anzahl benötigter Geschäftsapplikationen wird nicht zurückgehen und die Menge an verschiedenen Endgeräten wird tendenziell steigen. Folglich ist zu empfehlen, die Weiterentwicklung des Path-Framework fortzusetzen. Nebst den Verbesserungsvorschlägen, die aus den Usability Tests hervorgegangen sind, gibt es verschiedene denkbare Erweiterungsoptionen. Der visuelle Editor für das Erstellen von GUI's und das Einbinden eines Tools, das die Entwicklung des Backend, sprich die Geschäftslogik, unterstützt, sind nur zwei Ideen, die den Nutzen des Path-Framework noch weiter steigern würden.

Während dem Erarbeiten dieser Bachelorarbeit habe ich sehr viel gelernt und bin dankbar, dass ich eine Arbeit, bei der ein grosser Teil das Schreiben von Code ist, machen konnte. Man kann noch so viele Bücher über das Entwickeln von Software lesen, wirklich verstehen, um was es geht, kann man erst, wenn man sich direkt damit auseinandersetzt und probiert, etwas umzusetzen. Diese Bachelorarbeit ist definitiv eine Bereicherung für mich, sowohl vom Wissensstand her, als auch persönlich.

10 Literaturverzeichnis

Alpha Software. (o.J.). Mobile App Development Software Rated #1 By Developers | Alpha Software. Abgerufen von https://www.alphasoftware.com/

Berger, H.; Beynon-Davies, P. & Cleary, P. (2004). The Utility of a Rapid Application Development (RAD) Approach for a Large Complex Information Systems Development. *ECIS 2004 Proceedings,* 7. Abgerufen von http://aisel.aisnet.org/ecis2004/7/

Beynon-Davies, P., Mackay, H. & Tudhope, D. (2000). 'It's lots of bits of paper and ticks and post-it notes and things…': a case study of a rapid application development project. *Information Systems Journal, 10*, 195-216. doi:10.1046/j.1365-2575.2000.00080.x

Boehm, B. (1986). A Spiral Model of Software Development and Enhancement. *SIGSOFT Softw. Eng. Notes, 11*, 14-24. doi:10.1145/12944.12948

Franklin, C. (2016, 26. Oktober). Rapid Application Development: Know The Right Tools. Abgerufen von https://www.informationweek.com/software/enterprise-applications/rapid-application-development-know-the-right-tools/a/d-id/1327294

Hirschberg, M. (1998). Rapid Application Development (RAD): A Brief Overview. *Software Tech News, 2*, 1 & 7-8.

IBM. (2011). IBM Knowledge Center – Produktüberblick. Abgerufen von https://www.ibm.com/support/knowledgecenter/de/SSVRGU_8.5.3/com.ibm.designer.domino.ui.doc/wpd_overview.html

Martin, J. (1991). *Rapid application development.* Indianapolis, IN: Macmillan Publishing Co., Inc.

Mendix. (o.J.). Low-code Application Development Platform for Digital Transformation | Mendix. Abgerufen von https://www.mendix.com/

Metro-Bootstrap. (o.J.). metro-bootstrap: Twitter Bootstrap with Metro style. Abgerufen von https://talkslab.github.io/metro-bootstrap/components.html

Nnanna, J. (2017, 10. Oktober). Convert Your Angular Project To Mobile App Using Cordova. Abgerufen von https://medium.com/@nacojohn/convert-your-angular-project-to-mobile-app-using-cordova-f0384a7711a6

Omnis. (o.J.). Omnis Studio – Omnis Software Ltd. Abgerufen von https://www.omnis.net/

OutSystems. (o.J.). The #1 Low-Code Platform for Digital Transformation | OutSystems | OutSystems. Abgerufen von https://www.outsystems.com/

Ruparelia, N. (2010) Software Development Lifecycle Models. *ACM SIGSOFT Software Engineering Notes, 35*, 8-13. doi:10.1145/1764810.1764814

11 Anhang

Anhang 1: Fragebogen Usability Test

Usability Tests

Mit diesen Usability-Tests soll die Benutzerfreundlichkeit und die Verständlichkeit einer Path-Applikation abgeklärt werden. Hierzu wurde die Applikation 'Recipes' mit dem Path-Framework erstellt. Für den Test wird der Benutzer gebeten, sofern er/sie ein Android-Smartphone besitzt, sich die Recipes-Applikation herunterzuladen und zu starten. Dann soll sich der Tester zuerst einen Überblick über die App verschaffen und seinen/ihren allgemeinen Eindruck schildern. Danach wird er/sie gebeten verschiedene Szenarios (Use Cases) durchzuarbeiten. Während diesen Szenarien wird der Tester beobachtet und sein/ihr Vorgehen und Verhalten wird festgehalten. Zudem wird er geben laut zu denken während dem Vorgehen. Nach jedem Szenario soll eine kurze Reflexion durch den Benutzer gemacht werden. Schlussendlich wird der Tester nochmals nach seinen Eindrücken zu den soeben gemachten Erfahrungen befragt. Die Dauer eines Testes beträgt ca. 30 Minuten.

Szenarien

1. Schau dir das Rezept 'Spaghetti Bolognese' an. Dabei fällt dir auf, dass Mais als Zutat aufgelistet ist. Deiner Meinung nach gehört jedoch Mais bestimmt nicht in eine Bolognese-Sauce. **Lösche** aus diesem Grund die Zutat aus dem Rezept heraus.

2. Öffne das Rezept 'Hamburger'. Füge mindestens eine Zutat deiner Wahl (du kannst auch eine neue erstellen) dem Rezept 'Hamburger' hinzu.

3. Falls du in Szenario 2 keine eigene Zutat erstellt hast, hole dies nun nach.

4. Wieviel Rezepte stammen aus der Region Europa?

5. Wie viele Kalorien hat die Zutat Kokosnussmilch? (Hint: Der Button 'Zutat bearbeiten' sollte 'Zutat anzeigen' heissen)

6. In wie vielen Rezepten kommt die Zutat 'Zwiebeln' vor?

7. Wann hat die Zutat 'Eisbergsalat' Saison?

8. Erstelle ein neues Rezept. Falls du keine eigene Idee für ein Rezept hast, kann auch das Rezept 'Honigbrot' wie folgt hinzugefügt werden:

- Rezept: Honigbrot
  - Zubereitungszeit: 5min
  - Region: Europa
  - Rezept:

    1. Brot mit Butter bestreichen
    2. Brot mit Honig bestreichen

    En Guete!
- Zutaten
  - Brot (Saison: -, Kalorien: 265)
  - Butter (Saison: -, Kalorien: 717)
  - Honig (Saison: -, Kalorien 304)

Beobachtungsbogen

Alter: ☐ -25 ☐ 25-40 ☐ 41-55 ☐ 55+

Geschlecht: ☐ w ☐ m

Smartphone: _____

Technologiekenntnisse: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Beruflicher Hintergrund: _____

Allgemeiner Eindruck

Szenario 1

Szenario 2

Szenario 3

Szenario 4

Szenario 5

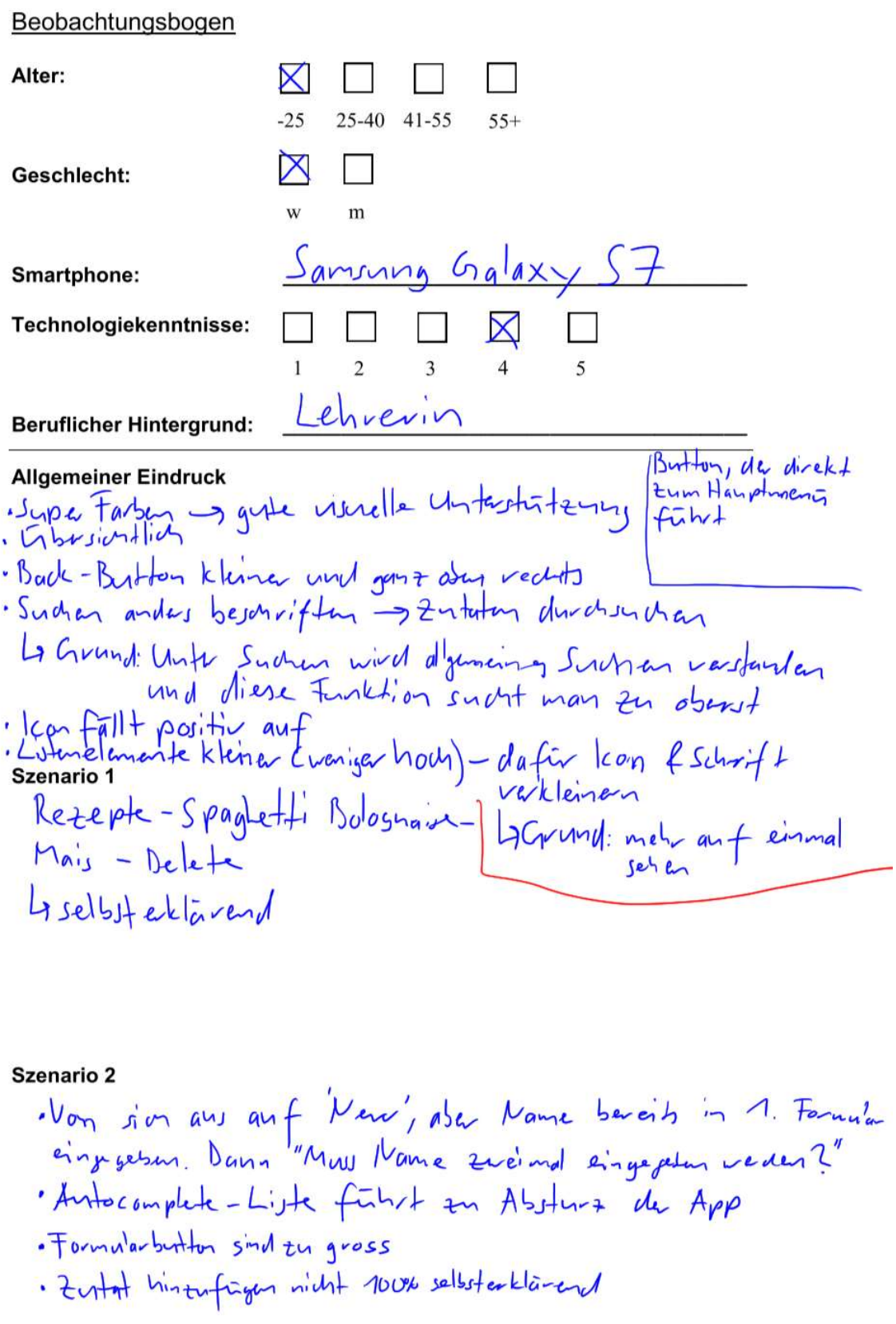
Szenario 6

Szenario 7

Szenario 8

Rückblick

Anhang 2: Ausgefüllter Beobachtungsbogen 1



Beobachtungsbogen

Alter: ☒ -25 ☐ 25-40 ☐ 41-55 ☐ 55+

Geschlecht: ☒ w ☐ m

Smartphone: Samsung Galaxy S7

Technologiekenntnisse: ☐ 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5

Beruflicher Hintergrund: Lehrerin

Allgemeiner Eindruck

- Supe Farben → gute visuelle Unterstützung
- Übersichtlich
- Back-Button kleiner und ganz oben rechts
- Suchen anders beschriften → Zutaten durchsuchen
  - ↳ Grund: Unter Suchen wird allgemein Suchen verstanden und diese Funktion sucht man zu oberst
- Icon fällt positiv auf
- Listenelemente kleiner (weniger hoch) – dafür Icon & Schrift verkleinern
  - ↳ Grund: mehr auf einmal sehen

Button, der direkt zum Hauptmenü führt

Szenario 1

Rezepte - Spaghetti Bolognaise – Mais – Delete

↳ selbsterklärend

Szenario 2

- Von sich aus auf 'Neu', aber Name bereits in 1. Formular eingegeben. Dann "Muss Name zweimal eingegeben werden?"
- Autocomplete-Liste führt zu Absturz der App
- Formularbutton sind zu gross
- Zutat hinzufügen nicht 100% selbsterklärend

- Beschriftung der Formular-Buttons müsste präziser sein
- Hintergrundfarbe für Formular einstellen können
- Formular 'Neue Zutat hinzufügen' ist klar
- Aber Verbindung Zutat-Rezept durch Form 'Zutat hinzufügen' ist nicht klar

Szenario 3

—

Szenario 4

Region-Europa → 1

intuitiv klar → sehr klar

Szenario 5

Zutaten - Durchgeswollt - Zutat anzeigen → 230

→ sehr klar

→ Suchfunktion als Durchsuchfunktion sehr praktisch

→ Sollte aber anders heißen (z.B. 'Zutaten durchsuchen')

→ Sollte auch ein wenig auffälliger gestaltet werden

→ Anderer Titel würde mehr ins Auge stechen

Soll benennbar sein

Szenario 6

Zutat - Suchfunktion - zwiebeln → 3

→ selbsterklärend

Szenario 7

Zutaten - Suchfunktion - Eingabemaske - Anzeige
Selbsterklärend
→ kürzester Weg

Szenario 8

- Rezept - Neues Rezept → Formular ausfüllen → soweit selbsterklärend
- "Wo sind Zutaten?" beim Ausfüllen → nicht klar wie Zutaten hinzugefügt werden können
- Nicht klar, dass Zutaten nach dem Erstellen des Rezepts noch hinzugefügt werden müssen.
- Zutaten sollen über Formular 'Rezept hinzufügen' hinzugefügt werden können → n-m Beziehung über Formular lösen können

Rückblick

- man findet sich gut zu recht
- alles sehr klar und verständlich
- Man sollte immer, wenn die App geöffnet wird im Hauptmenü starten → so ist die Orientierung einfach. Andererseits könnte es teils mühsam sein sich zu orientieren.
- Breadcrumb hat Titelfunktion und sollte deshalb markanter designed werden.
- Formulare anders designen → sieht nach 'Programmieren' aus → nur für Smartphones → Nutzerfreundlicher gestalten
- Oben links müsste Lupe zum Titel stehen, damit klar ist, dass dies als Homebutton benutzt werden kann

Verständlichkeit: g
Benutzerfreundlichkeit: g

Anhang 3: Ausgefüllter Beobachtungsbogen 2

Beobachtungsbogen

Alter: ☒ -25 ☐ 25-40 ☐ 41-55 ☐ 55+

Geschlecht: ☒ w ☐ m

Smartphone: Sony Xperia Z3

Technologiekenntnisse: ☐ 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5

Beruflicher Hintergrund: Studentin (Gesundheitswissenschaften)

Allgemeiner Eindruck

- Nativer Back-Button schließt App
- Recipes (Obm rechts) führt zum Absturz auf Smartphone
- Logout führt ebenfalls zum Absturz
- Übersichtlich
- Verschiedene Einstiegswege
- Suchfunktion ist aufgefallen

Szenario 1

- Rezepte → raus/öffnen ist klar
- Nicht klar, dass gescrollt werden kann
- Titel für Listen (Liste der Zutaten)

Szenario 2

- initiv richtig vorgeg
- Autocomplete schützt ab
- 'New' ist nicht klar

Szenario 3

–

Szenario 4

Region - Europa

↳ Liste von Zutaten unten ist speziell → nicht in Rezept anzeigen

Szenario 5

Zutat – Zutat anzeigen → durchgestrichen, da Suchfunktion im Kopf

Szenario 6

Zutat – Zwischen (Suchfunktion) → Liste der Rezepte nicht intuitiv

Szenario 7

–

Szenario 8

- Rezept anlegen ist selbsterklärend
- Es ist nicht selbsterklärend, dass Zutaten zum Rezept hinzugefügt werden müssen

Rückblick

- Handlich
- Schnell zum Sachen hinzuzufügen
- Alert: Zutaten hinzufügen nach Erstellen von Rezepten
- Immer gleich aufgebaut
- Farben helfen zu verstehen

Benutzerfreundlichkeit: 7
Verständlichkeit: 8

Anhang 4: Ausgefüllter Beobachtungsbogen 3

Beobachtungsbogen

Alter: ☐ -25 ☒ 25-40 ☐ 41-55 ☐ 55+

Geschlecht: ☐ w ☒ m

Smartphone: Samsung Galaxy S8

Technologiekenntnisse: ☐ 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5

Beruflicher Hintergrund: Hauswart

Allgemeiner Eindruck

- Klar, Verständlich
- Gute Funktion (Region)

→ Back-Button (nativ)

Szenario 1

Rezepte - Zutaten - Delete

→ problemlos

Szenario 2

Zutat, die noch nicht existiert → nicht intuitiv

ansonsten gut

Szenario 3

—

Szenario 4

Region - Europa → 1 → intuitiv

Szenario 5

- ohne Suchfunktion

→ gefunden ohne Probleme

→ Suchfunktion ist aufgefallen
→ Suchfunktion problemlos bedient
→ Suchfunktion auffällig

Szenario 6

Zutaten - Zutat → intuitiv

Szenario 7

intuitiv richtig

Szenario 8

Zutat angelegt → direkt wieder gefunden
↳ intuitiv

→ Rezept - Menü Rezept

! → selbsterklärend App ist

→ Zutaten hinzufügen von sichtbar - könnte aber verbessert sein

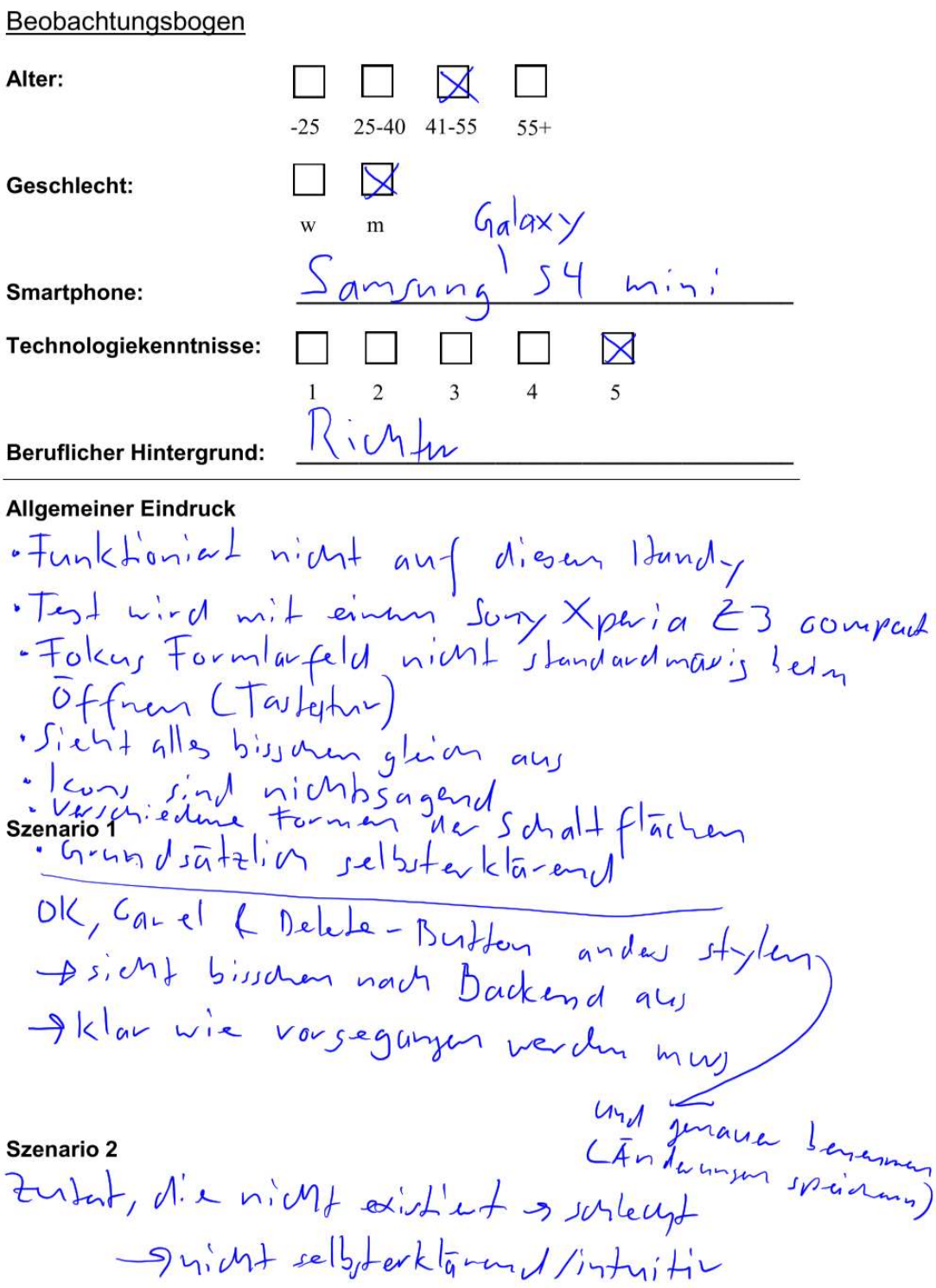
Rückblick → Zutat hinzufügen (neu) beim 2. Mal direkt über "Neu"

- sehr simpel (einfache Bedienbarkeit)
- Back / Search auch auf Desktop → Translation-Service

Benutzerfreundlichkeit: 8

Verständlichkeit: 8

Anhang 5: Ausgefüllter Beobachtungsbogen 4



Beobachtungsbogen

Alter: ☐ -25 ☐ 25-40 ☒ 41-55 ☐ 55+

Geschlecht: ☐ w ☒ m

Smartphone: Samsung Galaxy S4 mini

Technologiekenntnisse: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☒ 5

Beruflicher Hintergrund: Richter

Allgemeiner Eindruck

- Funktioniert nicht auf diesem Handy
- Test wird mit einem Sony Xperia E3 compact
- Fokus Formularfeld nicht standardmäßig beim Öffnen (Tastatur)
- Sieht alles bissken gleich aus
- Icons sind nichtssagend
- Verschiedene Formen der Schaltflächen
- Grundsätzlich selbsterklärend

Szenario 1

OK, Cancel & Delete-Button anders stylen
→ sieht bisschen nach Backend aus
→ klar wie vorgegangen werden muss

und genaue bezeichnen (Änderungen speichern)

Szenario 2

Zustand, die nicht existiert → schlecht
→ nicht selbsterklärend / intuitiv

Szenario 3

–

Szenario 4

Regionen - Europa → intuitiv

Szenario 5

→ intuitiv → ohne Suchfunktion

→ Suchfunktion geht unten

↳ Lupe in Search oder in Input feld

→ Suchfunktion die alles durchsucht

Szenario 6

Intuitiv → ohne Suchfunktion

Szenario 7

→ Intuitiv

Szenario 8

→ Autocomplete lässt App abstürzen
→ Zuletzt vor sich aus hinzugefügt
→ Neu → intuitiv wird Name eingegeben und dann auf Neu geklickt, wo Name nochmals eingegeben werden muss. → Name in neues Formular schreiben.

Rückblick

- Verschiedene Wege kann ans Ziel zu gelangen
- Standard gut → Intuitiv

Benutzerfreundlichkeit: 8
Verständlichkeit: 7

Anhang 6: Ausgefüllter Beobachtungsbogen 5

Beobachtungsbogen

Alter: ☒ -25 ☐ 25-40 ☐ 41-55 ☐ 55+

Geschlecht: ☐ w ☒ m

Smartphone: Samsung Galaxy S5

Technologiekenntnisse: ☐ 1 ☐ 2 ☒ 3 ☒ 4 ☐ 5

Beruflicher Hintergrund: Automech / Student

Allgemeiner Eindruck

- Verschiedene Eingabewege
- Ein paar 4 übersichtlich
- Für Fingerspitzen geeignet

Szenario 1

Selbsterklärend & intuitiv richtig vorgegangen

Szenario 2

Rezepte - Hamburgern - Rezept hinzufügen,
→ Von sich aus auf 'Neu'
→ Erst ein paar eingeben

Szenario 3

selbsterklärend

Szenario 4

intuitiv verständlich

Szenario 5

Zutaten-Suchfunktion → Zutat anzeigen
→ intuitiv klar

→ Suchfunktion gesehen
→ aber geht unter

Szenario 6

Zutaten-Suchen Liste unten
→ selbsterklärend

→ Button Zutat anzeigen → Details benannt
Lymon ist bereits
in der Zutat

Szenario 7

–

Szenario 8

Rezept – Neues Rezept
→ intuitiv verständlich
→ nicht klar, dass Zutaten neu hinzugefügt werden müssen
→ Zutaten direkt über neues Rezept hinzufügen

Rückblick

- Sehr übersichtlich
- Rezepte & Zutaten kann verwirrend sein
- Rezept soll nicht gleich bearbeitbar sein beim anlegen

Benutzerfreundlichkeit: 9
Verständlichkeit: 7

Anhang 7: Ausgefüllter Beobachtungsbogen 6

Beobachtungsbogen

Alter: ☐ -25 ☒ 25-40 ☐ 41-55 ☐ 55+

Geschlecht: ☐ w ☒ m

Smartphone: Samsung Galaxy S8

Technologiekenntnisse: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☒ 5

Beruflicher Hintergrund: Student Wirtschaftsinformatik

Allgemeiner Eindruck

- Back-Button nativ oder zumindest Warnung, dass App verlassen wird
- Übersichtlich
- reagiert gut
- gut lesbar
- Selbsterklärend

Szenario 1

Rezepte - Bolognese - Mail - Delete

→ klar

Szenario 2

Rezepte - Hamburger - Zutat hinzufügen

→ Menü ist verwirrend
Symbol klar, wie Zutat, die noch nicht existiert

→ Nach Durchführung selbst verstanden wie aufgebaut, ist aber unnötig kompliziert

→ Ok-Button ist vorhanden, wenn ein bereits existierender Zucht ausgewählt wird. Sonst ist dort der Neu-Button.
↳ Name steht bereits im Formular 'Neue Zucht'

Szenario 3

–

Szenario 4

klar

Szenario 5

klar, wenn Button richtig angesprochen ist

→ Suchfunktion ist nicht aufgefallen
↳ geht unter
↳ sollte Durchsuchen heißen

Szenario 6

klar

Szenario 7

–

Szenario 8

Rezept - Neues Rezept → soweit klar
→ klar, dass Zutaten hinzugefügt werden müssen
↳ nicht intuitiv verständlichste Lösung

→ automatisch Zutatenfeld für Kalorien

Rückblick

- Intuitiv
- Übersichtlich
- Einfache Bedienbarkeit

Verständlichkeit: 9
Benutzerfreundlichkeit: 9.5

Anhang 8: Ausgefüllter Beobachtungsbogen 7

Beobachtungsbogen

Alter: ☒ -25 ☐ 25-40 ☐ 41-55 ☐ 55+

Geschlecht: ☐ w ☒ m

Smartphone: Samsung Galaxy S8

Technologiekenntnisse: ☐ 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5

Beruflicher Hintergrund: Hauswart

Allgemeiner Eindruck

- Back-Button → nativ
- Alle Zutaten gleiches Icon (evtl. Icon für Zutaten-Kategorien)

Szenario 1

intuitiv

Szenario 2

Zutat, die noch nicht vorhanden ist hinzufügen ist nicht intuitiv

Szenario 3

–

Szenario 4

intuitiv

Szenario 5

intuitiv

→ Suchfunktion geht auch
↳ in Menü nehmen

Szenario 6

Wie Suchfunktion → intuitive Vorgehen

Szenario 7

Intuitiv, ohne Suchfunktion

Szenario 8

Rezept - Rezept hinzufügen
→ intuitiv
→ Zutaten nicht von hinzugefügt
→ Neue Zutat, die noch nicht existiert, beim 2. mal richtig vorgegangen

Rückblick

- Suchfunktion oben oder Menü
- Unterteilung Zutaten in Kategorien
- Filter raster
- Zutaten auswählen können, die man zuhause hat → Menüs angeben

Benutzerfreundlichkeit: 8
Verständlichkeit: 9

Anhang 9: Ausgefüllter Beobachtungsbogen 8

Beobachtungsbogen

Alter: ☒ -25 ☐ 25-40 ☐ 41-55 ☐ 55+

Geschlecht: ☐ w ☒ m

Smartphone: Samsung Galaxy S7 Edge

Technologiekenntnisse: ☐ 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5

Beruflicher Hintergrund: Detailhandel / Student

Allgemeiner Eindruck

- schnell gemerkt wo was ist

Szenario 1

Rezept über Regionen

→ problemlos gelöscht

Szenario 2

→ verschiedene Wege führen zum Ziel

→ 2. Anlauf neue Zutat (new) gefunden

→ zuerst direkt Namen angesehen

Szenario 3

Regionen - Europa → 1

Szenario 4

Gescrollt → Gefunden (ohne Suchfunktion)
Suchfunktion anders platzieren (wird übersehen)
↳ anders gestalten

Szenario 5

Über Suchfunktion (Zutaten)

Szenario 6

Szenario 7

Über Suchfunktion
Zutat unter Rezepte gesucht, schnell gemerkt,
dass Zutat, als nichts gefunden wurde

Szenario 8

Rezepte – Neues Rezept

→ Pizza Prosciutto et Funghi
↳ Detail problemlos ausgefüllt / Beschreibung war nicht klar
↳ Zutaten dort hinzufügt
→ Zutaten von sich aus hinzugefügt

Rückblick

→ Zutat hinzufügen (Rezept)
↳ 2 Felder → 1 bestehende Zutat hinzufügen
1 neue Zutat anlegen und hinzufügen
→ Gruppen (UA)
→ Back-Button nativ
Benutzerfreundlichkeit: 9
Verständlichkeit: 8

Anhang 10: Ausgefüllter Beobachtungsbogen 9

Beobachtungsbogen

Alter: ☐ -25 ☒ 25-40 ☐ 41-55 ☐ 55+

Geschlecht: ☒ w ☐ m

Smartphone: Samsung Galaxy S7

Technologiekenntnisse: ☐ 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5

Beruflicher Hintergrund: Technik

Allgemeiner Eindruck

Back-Button nativ
mehrheitlich selbsterklärend

Szenario 1

Rezepte-Zutaten - Mais - Delete
--> selbsterklärend
Hover-Effekt für Smartphone-Version entfernen
Unterschiedlich Symbole in den Listen/Unterschiedliche Farben

Szenario 2

Rezept-
--> Selbst gemerkt wie eine Zutat, die noch nicht existiert, hinzugefügt werden kann
Saison nicht als required Feld
Zutat hinzufügen - wenn sie noch nicht existiert in autocomplete-liste 'new' anzeigen

Szenario 3

-

Szenario 4

Region-Europa --> 1
intuitiv verständlich

Szenario 5

Suchfunktion direkt aufgefallen und benutzt
Zutaten-Suchfunktion-Kokosnussmilch
intuitiv verständlich

Szenario 6

Zutaten-Zwieblen--> in allen
intuitiv verständlich

Szenario 7

Zutaten - Suchfunktion - Zutat bearbeiten

Szenario 8

Rezepte-Neues Rezept
Von sich aus Zutaten hinzugefügt

Rückblick

verschiedene Icons und Farben in einer Liste (Bring! App)
klar und selbsterklärend
intuitiv richtig vorgegangen
Suchfunktion auffälliger gestalten, damit sie direkt ins Auge sticht
Suchfunktion neben Backbutton
klobig --> Baustein-Satz; dafür ist es sehr einfach zu bedienen und verständlich
Backbutton neben Titel und kleiner --> mehr Platz und bessere Übersicht

Benutzerfreundlichkeit: 6
Verständlichkeit: 9

Anhang 11: Ausgefüllter Beobachtungsbogen 10

Beobachtungsbogen

Alter: ☐ -25 ☐ 25-40 ☐ 41-55 ☒ 55+

Geschlecht: ☒ w ☐ m

Smartphone: Samsung 7

Technologiekenntnisse: ☐ 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5

Beruflicher Hintergrund: Lehrerin / Schulleiterin

Allgemeiner Eindruck

- Gleich aufgebaut → Konsistenz
- Intuitiv verständlich
- Selbsterklärend

Szenario 1

- Zutaten nicht auf Anhieb gefunden
- Sprache ist nicht Konsistenz

Szenario 2

- Nicht vorhandene Zutat hinzufügen ist nicht selbsterklärend!

Szenario 3

—

Szenario 4

Region - Europa → 1

→ intuitiv

Szenario 5

Zustehen – Kollaborativ gesucht ohne Suchfunktion

Szenario 6

Zustehen – Zuviel

→ überall

→ Offensichtlich

Szenario 7

Intuitiv richtig

Szenario 8

- Intuitiv richtig navigiert
- Nicht auf Anhieb gefunden, wie Zutaten hinzugefügt werden

Rückblick

- Nicht offensichtlich, dass Scrollen möglich ist
- Feld 'Rezept' sollte Zubereitung heissen
- Autocomplete-Feld stürzt ab auf 'App'. Im Browser funktioniert es.
- Sehr einfach handzuhaben
- Grosse Schaltflächen (+)
- Neue Zutat nur Name required
- Symbole und Farben sind ansprechend

Benutzerfreundlichkeit: 10

Verständlichkeit: 8

Anhang 12: Anleitung Cordova im Gihub Wiki

Converting a Path Web Application to a Mobile App

Edit | New Page

zhaw-weberm16 edited this page 6 minutes ago · 9 revisions

This tutorial shows how you can convert your Path web application to a mobile app, which can be deployed on any mobile operating system. To execute this tutorial you have to meet the following requirements:

- Path Application with Backend
- Cordova CLI installed
- Angular CLI installed

***Important**: Before you execute this tutorial you should ensure that you have a backup of your application*

1. Cordova create

The first step is to create a new Cordova project. Navigate to to desired directory and execute the following command:

```
C:\pathProject\Beispielapplikationen>cordova create recipesCordova com.herokuapp.pathRecipes RecipesCordova
Creating a new cordova project.

C:\pathProject\Beispielapplikationen>
```

recipesCordova in this example is the name of the folder in which the Cordova project should be created. com.herokuapp.pathRecipes is a reverse domain-style identifier and RecipesCordova is the title of the project. Please refere to Documentation Apache Cordova for more details.

2. Cordova platform add

The next step is to add the desired platforms to the Cordova project. Please navigate into the directory of your Cordova project and execute the following command:

```
C:\pathProject\Beispielapplikationen\recipesCordova>cordova platform add android

C:\pathProject\Beispielapplikationen\recipesCordova>cordova platform add browser
```

Important: You have to install the SDK for every platform you want to target (exception: browser)!

3. Cordova requirements

To check whether you meet the requirements of each platform you can run the following command:

```
C:\pathProject\Beispielapplikationen\recipesCordova>cordova requirements
Android Studio project detected

Requirements check results for android:
Java JDK: installed 1.8.0
Android SDK: installed true
Android target: installed android-27,android-26
Gradle: installed C:\Program Files\Android\Android Studio\gradle\gradle-4.4\bin\gradle
```

Pages 1

Converting a Path Web Application to a Mobile App

+ Add a custom sidebar

Clone this wiki locally

https://github.com/zhaw-web

Clone in Desktop

You may get the following error for the android platform:

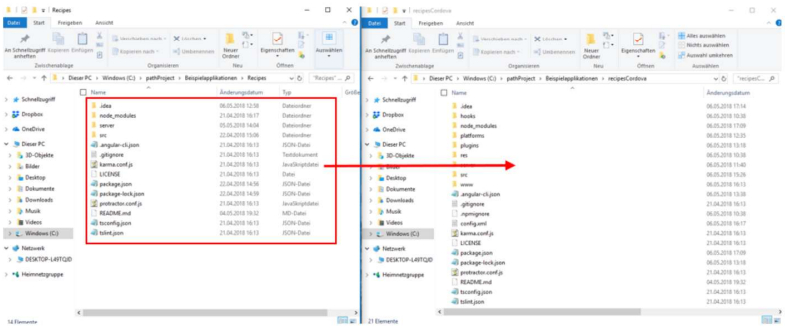
```
C:\pathProject\Beispielapplikationen\recipesCordova>cordova requirements
Android Studio project detected

Requirements check results for android:
Java JDK: installed 1.8.0
Android SDK: installed true
Android target: not installed
Please install Android target / API level: "android-26".

Hint: Open the SDK manager by running: "C:\Users\Manu\AppData\Local\Android\sdk\tools\android.bat"
You will require:
1. "SDK Platform" for API level android-26
2. "Android SDK Platform-tools (latest)
3. "Android SDK Build-tools" (latest)
Gradle: installed C:\Program Files\Android\Android Studio\gradle\gradle-4.4\bin\gradle
(node:15924) UnhandledPromiseRejectionWarning: CordovaError: Some of requirements check failed
```

If so, you can solve this problem by executing the following steps:

1. Open Android Studio
2. Click on Configure - SDK Manager
3. Check the option 'Android 8.0 (Oreo), API level 26'
4. Click on Apply
5. Run 'cordova requirements' again

4. Merge your Path project with the Cordova project

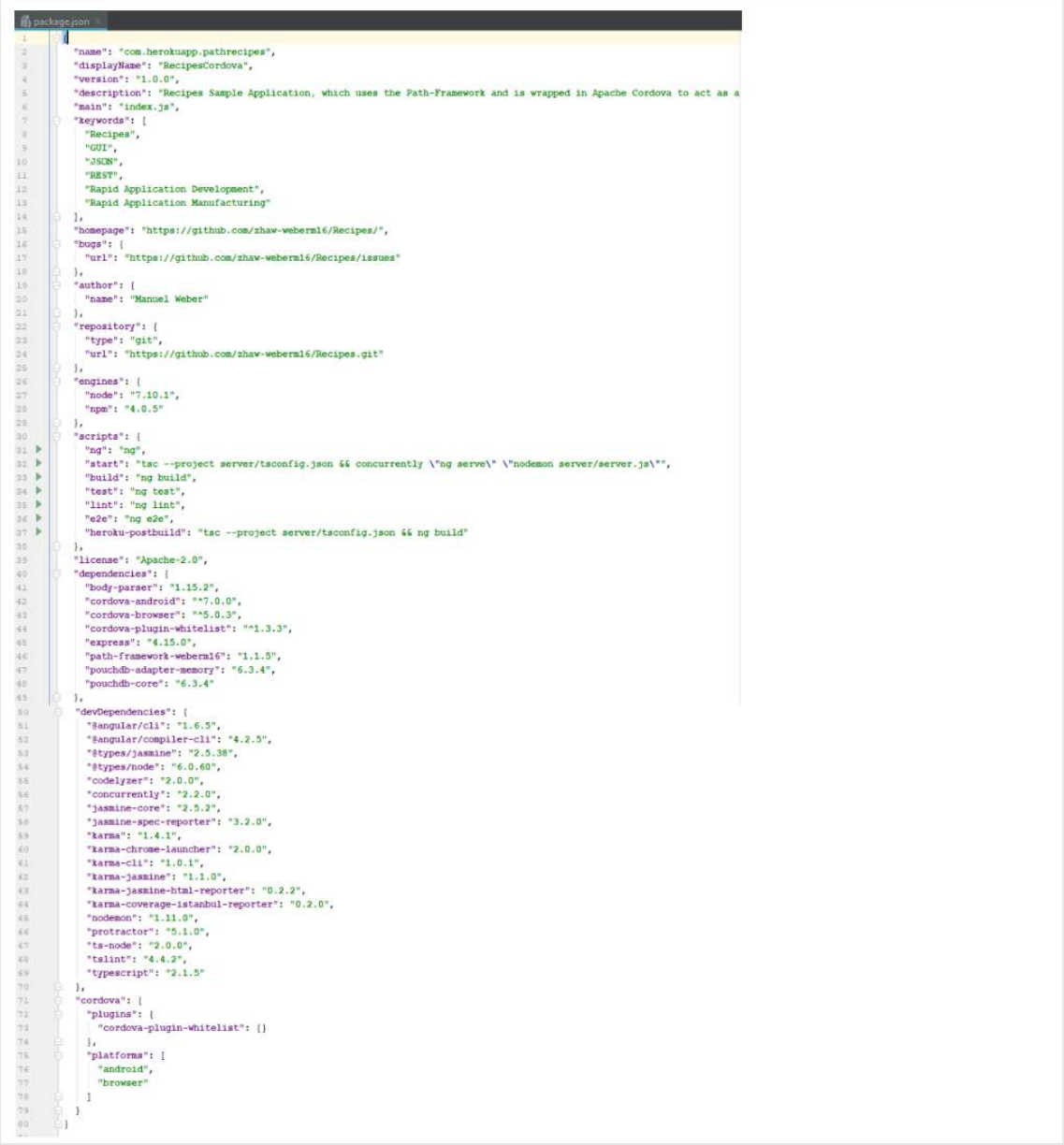
To merge these two projects copy all folders and files, except the package.json file, from your Path project root directory to the Cordova project root directory.



Hint: You don't have to copy the node_modules folder

5. Merge the package.json files

Open both package.json files and merge them carefully. The resulting package.json file (in the Cordova project) should look like the one below.



```
package.json
{
  "name": "com.herokuapp.pathrecipes",
  "displayName": "RecipesCordova",
  "version": "1.0.0",
  "description": "Recipes Sample Application, which uses the Path-Framework and is wrapped in Apache Cordova to act as a
  "main": "index.js",
  "keywords": [
    "Recipes",
    "GUI",
    "JSON",
    "REST",
    "Rapid Application Development",
    "Rapid Application Manufacturing"
  ],
  "homepage": "https://github.com/zhaw-weberm16/Recipes/",
  "bugs": {
    "url": "https://github.com/zhaw-weberm16/Recipes/issues"
  },
  "author": {
    "name": "Manuel Weber"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/zhaw-weberm16/Recipes.git"
  },
  "engines": {
    "node": "7.10.1",
    "npm": "4.0.5"
  },
  "scripts": {
    "ng": "ng",
    "start": "tsc --project server/tsconfig.json && concurrently \"ng serve\" \"nodemon server/server.js\"",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "heroku-postbuild": "tsc --project server/tsconfig.json && ng build"
  },
  "license": "Apache-2.0",
  "dependencies": {
    "body-parser": "1.15.2",
    "cordova-android": "^7.0.0",
    "cordova-browser": "^5.0.3",
    "cordova-plugin-whitelist": "^1.3.3",
    "express": "4.15.0",
    "path-framework-weberm16": "1.1.5",
    "pouchdb-adapter-memory": "6.3.4",
    "pouchdb-core": "6.3.4"
  },
  "devDependencies": {
    "@angular/cli": "1.6.5",
    "@angular/compiler-cli": "4.2.5",
    "@types/jasmine": "2.5.38",
    "@types/node": "6.0.60",
    "codelyzer": "2.0.0",
    "concurrently": "2.2.0",
    "jasmine-core": "2.5.2",
    "jasmine-spec-reporter": "3.2.0",
    "karma": "1.4.1",
    "karma-chrome-launcher": "2.0.0",
    "karma-cli": "1.0.1",
    "karma-jasmine": "1.1.0",
    "karma-jasmine-html-reporter": "0.2.2",
    "karma-coverage-istanbul-reporter": "0.2.0",
    "nodemon": "1.11.0",
    "protractor": "5.1.0",
    "ts-node": "2.0.0",
    "tslint": "4.4.2",
    "typescript": "2.1.5"
  },
  "cordova": {
    "plugins": {
      "cordova-plugin-whitelist": {}
    },
    "platforms": [
      "android",
      "browser"
    ]
  }
}
```

6. Run npm install

Run npm install to install all dependencies for your project.

7. Run ng serve for a development server

By now you have successfully converted your Path application to a mobile app. You can run 'ng serve' for a development server. Navigate to http://localhost:4200/. The app will automatically reload if you make any changes.

8. Add the Cordova plugin

To add the Cordova plugin you have to add the following script tag to your index.html file:

```
<script type="text/javascript" src="cordova.js"></script>
```

Hint: cordova.js does not have to exist yet, it will be generated if you build the application

Add the Cordova plugin with the command below:

```
cordova plugin add cordova-plugin-device
```

Now you can add other Cordova plugins to access native functions of smartphones like the camera or GPS.

9. Build and run your app

To build and run your app you have to make some more adjustments to you app:

1. Open the index.html file and change the tag `<base href="/">` to `<base href="./">`. This enables Cordova to access files in a directory path since we are not hosting on a server.
2. A angular project creates a folder called 'dist' during the build process. Cordova does not know this folder. Instead it uses a folder called 'www'. This means that we have to change to output for our project from the 'dist' to the 'www' folder. To achieve this, open the .angular-cli.json file and change 'outDir' to 'www'.
3. Run 'ng build'. This will create the 'www' folder with the required output files.

10. Test your application in an emulator

Now you can test your application in an emulator. To create an Android emulator you can use Android Studio by executing the following steps:

1. Launch the AVD (Android Virtual Device) Manager from Android Studio.

2. Create a new virtual device.
3. Configure your device.
4. Run 'cordova emulate android' to emulate your app in the create emulator.

11. Publish your app

Now you can go on and publish your app on the Google Play Store or the Appstore of Apple if you like to.